Componentes

Un programa en C++ está definido por funciones (grupo de instrucciones que pueden o no hacer algún cálculo), donde la función

principal debe ser llamada **main**. La composición general de un programa en C++ es:

- directivas de preprocesamiento.
- declaración globales.
- funciones definidas por el usuario
- función main
- comentarios para entender el funcionamiento del programa

Directivas

Las directivas de preprocesamiento son definidas para que el compilador realice algo antes de compilar el programa (revisar si la

sintaxis es correcta y generar un código ejecutable por la computadora), como el incluir funciones de alguna biblioteca conocida

en particular.

En C la directiva para incluir bibliotecas es como las siguientes:

Ejemplos:

#include <iostream.h>

Esta directiva permitirá utilizar la función de lectura **cin** y la función de escritura **cout** propias de C++.

#include <stdio.h>

Esta directiva permitirá utilizar las funciones de lectura y escritura propias de C.

Estas directivas también funcionan con C++, aunque existen algunos compiladores que no las

soportan en cuyo caso es necesario

hacer uso de los namespaces.

Un namespace es una declaración de región. El propósito de estos es localizar el nombre de identificadores para evitar colisiones.

Para tener compatibilidad, cuando un programa en C++ incluye una libreria de C, haremos uso del namespace.

Ejemplo:

En C tenemos #include <iostream.h>

con el respectivo en C++ #include <iostream>
using namespace std;

Declaraciones Globales

Las declaraciones globales son definiciones de variables o constantes que serán utilizadas por cualquiera de todas las funciones definidas en el programa.

La sintaxis para la declaración de una **variable** es:

Tipo var1, var2,..., var3;

o bien

Tipo var1= valor, var2,..., var3;

Ejemplos:

int edad;

Esta declaración indica que la variable edad definida como un valor entero puede ser utilizada en cualquier función.

double venta = 0.0;

Esta declaración presenta la declaración de una variable como valor doble a la cual se le da un valor inicial y puede ser usada en cualesquier función.

char nombre[30];

Esta declaración es para declarar una variable de tipo cadena de caracteres.

La sintaxis para la declaración de una **constante** es:

const double PI = 3.1415;

Esta declaración representa el valor de la constante PI la cual podrá utilizarse en cualquier

función.

Reglas para nombre de variables

- Deben empezar con letra o underscore
- Pueden contener letras, números y underscore
- Las letra mayúsculas y minúsculas son diferentes para C++

Funciones definidas por el usuario y función main

La definición de una función se realiza escribiendo primero el tipo del valor de retorno de la función, después el nombre de la función, posteriormente entre paréntesis las variables que utilizará dicha función (parámetros) y finalmente las instrucciones de la función. Ejemplos:

```
double promedio( int a, int b, int c)

{
  return (a + b + c ) / 3.0;
}
```

Declara a la función promedio, la cual recibe tres valores enteros y calcula y regresa el promedio de ellos.

```
int  main ()

{
      cout << " Función main desplegando un mensaje ";

   return 0;
}
```

Declara a la función principal main, no tendrá parámetros, ni regresará algún valor.

A través de las funciones cin y cout se pueden tomar datos del teclado y desplegar información correspondientemente en la computadora, dichas funciones pertenecen a la biblioteca iostream.h

Comentarios

Los comentarios pueden escribirse utilizando alguna de las siguientes formas:

1. comentando en la misma linea, utiliza //, ejemplo

int edad; // la edad se utilizará como un valor entero.

2. comentando entre varias lineas, utiliza /* */, ejemplo:

/* La siguiente funcion promedio recibe tres valores enteros

y calcula y regresa el promedio de los tres a través

de un valor real */

NOTA IMPORTANTE

Todos los ejemplos de este curso están basados en el compilar Borland TurboC++ sugerido.

Ejemplos de Programas

A continuación tendrás algunos ejemplos de programas en C++:

Ejemplo I

```
#include <iostream.h>  // biblioteca para el manejo de lectura y escritura

int edad;
int main()
{
        cout  <<  "Da  la  Edad  ";                          //  se  despliega  el
mensaje a la pantalla
        cin  >>  edad;                                  //  se  lee  el  dato  del
teclado (la edad)
       if  (edad  >=  18)                              //  validando  si  la  edad
es mayor o igual a 18
                cout  <<  "Eres  mayor  de  Edad";         //  "Mayor  de  Edad"  es
desplegado
       else
                cout  <<  "Eres  menor  de  Edad";         //  "Menor  de  Edad"  es
desplegado
    return 0;
}
```

Ejemplo II

```
#include <iostream.h>

/* Declaración de variables globales y funciones.
En este ejemplo no hay variables globales, solo
declaramos la función que calculará el promedio de
3 calificaciones que son leídas en la función main
y enviadas a la función promedio */

/* Funcion promedio que recibe tres datos y
regresa el cálculo del promedio de los tres */

double promedio(int a, int b, int c)
{
return (a + b + c ) / 3.0;
}


int main()
{
int edad1, edad2, edad3; // declaramos las variables
como locales
cout << "Da la Edad 1";
cin >> edad1;
cout << "Da la Edad 2";
cin >> edad2;
cout << "Da la Edad 3";
cin >> edad3;
cout << "El promedio de las 3 edades es = " << promedio (edad1,
edad2, edad3);
return 0; /* se envían como
parámetro
}


</SPAN< p>
```

Ejercicio

Baja el compilador de Borland turboC++ en la dirección:
http://www.borland.com/products/downloads/download_cbuilder.html#

Baja el editor de Crimson en la siguiente dirección:
http://www.crimsoneditor.com/english/download.html

Instala el compilador dando doble clic al archivo de instalación: **freecommandLinetools.exe**

Baja de los recursos los archivos BCC.BAT y RUN.BAT y copialos en la ruta
c:\Borland\BCC55\Bin

Instala el editor Crimson dando doble clic al archivo de instalación **crimsoneditor.exe**

Configura el editor de Crimson usando los siguientes pasos:

- Ejecuta el editor, al hacerlo por primera vez aparece un mensaje de error, solo da Aceptar
- En el Menú ***Tools***, selecciona ***Preferences***
- En la ventana de Categorías, selecciona ***Syntax*** y elije ***C/C++***
- En la misma ventana, selecciona ahora ***User Tools*** y has clic donde dice ***Empty Ctrl 1*** y llena los recuadros de la siguiente forma:
  Menú Text : Compilar (puedes poner cualquier texto, es solo el mensaje que aparecerá en la ventana de Tools, que indica la opción de compilar el programa).
  Command: *c: \borland\bcc55\bin\bcc.bat*
  Argument: haciendo clic en la flecha selecciona *File Name*
  Initial dir: haciendo clic en la flecha selecciona *File Dir*
  Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.

  Ahora has clic donde dice ***Empty Ctrl 2*** y llena los recuadros de la siguiente forma:
  Menú Text : Ejecutar (puedes poner cualquier texto, es solo el mensaje que aparecerá en la ventana de Tools, que indica la opción de ejecutar el programa).
  Command: *c: \borland\bcc55\bin\run.bat*
  Argument: haciendo clic en la flecha selecciona *File Title*
  Initial dir: haciendo clic en la flecha selecciona *File Dir*
  Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.

  Por útlimo da Apply.

Prueba cuidadosamente los ejemplos, revisando que sucede con diferentes valores.

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Tipos de Datos

Los tipos de datos que maneja C++ son :

- Enteros

  Dentro de los enteros están los tipos: short, int, long

  Los cuales varían en rango de acuerdo al compilador que se utilice, siendo long el de rango mayor y short el menor.

- Flotantes

  Dentro de los flotantes C++ tiene los tipos: float, double y long double, donde al igual que los enteros varía el rango de cada

  uno de acuerdo al compilador que se utilice. De igual forma el float es el de menor rango siendo long double el de rango

  mayor.

- Caracteres

  Se utiliza el tipo char. Para representar un caracter en C++ se utilizan apóstrofes.

  Ejemplos: ´a´, ´b´, ´5´

  Para representar una cadena de caracteres se utilizan las comillas.

  Ejemplo: "Soy una cadena"

Estatuto de Lectura

Sintaxis del estatuto **cin**

cin >> Lista de variables;

donde Lista de variables puede ser una o más variables separadas por >>

Ejemplos:

cin >> a >> b; // estamos esperando la entrada de dos valores del tipo definido en la

declaración de a, y b

cin >> num; // estamos esperando la entrada de un solo valor

Estatuto de Escritura

Sintaxis del estatuto **cout**

cout << Lista;

donde Lista puede ser variables, valores constantes o expresiones separadas por <<

Ejemplos:

cout << "Dame el numero "; // estamos desplegando el mensaje en pantalla

cout << "El resultado es " << resul; // estamos desplegando el letrero y el valor de la variable resul

Recordemos que para poder utilizar el estatuto cin y cout se requiere la directiva:

#include <iostream.h>

Existen caracteres especiales que pueden ser usados con el cout dentro de comillas. Entre ellos están:

- \n realiza un cambio de línea equivalente a usar endl
- \t tabulador
- \ a sonido de la campana
- \ " comilla doble
- \ ' comilla simple

Ejemplos:

cout << "Nombre " << "\t" ; // despliega el mensaje Nombre, realiza un Tab
// ahí quedará el cursor

cout << "El resultado es " << resul << endl; //Despliega el letrero, el valor de la variable resul y
//brinca a la siguiente línea donde quedará el cursor

Operador de Asignación

Sintaxis:

Variable = valor;

donde valor puede ser una constante, una variable o una expresión.

En C++ es válido usar el siguiente operador de asignación

a = b = 7;

el cual se evalúa de izquierda a derecha.

Ejemplos de Programas

A continuación tendrás algunos ejemplos de programas en C++:

Ejemplo I

```
#include <iostream.h>

/* Declaración de variables globales */

      double largo, ancho;           // declaramos las variables como dobles


 int main()
   {
   cout << "Dame el largo "<< endl;
   cin >> largo;
   cout << "Dame el ancho " << endl;
   cin >> ancho;
   cout << "El area es" << largo * ancho;  //  área de un rectángulo
   return 0;
   }
```

Ejemplo II

```
#include <iostream.h>

/* Declaración de variables globales */

      double centimetro;
      int pulgada;

int main()
   {
   cout << "Dame las pulgadas "<< endl;
   cin >> pulgada;
   centimetro = pulgada * 2.54;
   cout << pulgada <<" pulgadas corresponden a " << centimetro <<" centimetros"<<
endl;
```

```
return 0;
}
```

Ejercicio

Realiza los siguiente programas en la herramienta de C++ que seleccionaste.

1. Un programa que calcule el área de un triángulo dadas la base y altura, donde Area = base * altura ver solución
2

2. Escribir un programa que calcule la longitud y el área de una circunferencia. ver solución

longitud de la circunferencia = 2 * PI * radio
área de la circunferencia = PI * radio2

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Operadores Aritméticos

Operador en C++	Significado
+	suma
-	resta
*	multiplicación
/	división
%	residuo

Todos los operadores que se muestran en esta tabla son binarios; es decir, trabajan con dos operandos.

Los operadores **+**, **-** y ***** funcionan de la manera conocida.

El operador **/** funciona de diferente manera si trabaja con datos de tipo entero o de tipo flotante. Con datos de tipo flotante

funciona de la manera tradicional; pero al realizarse una división entre dos números enteros, el operador **/** regresa el cociente de

la división entera; es decir, regresa la parte entera del resultado (si hay fracción la elimina).

Por ejemplo:

2/3 da como resultado 0

pero

2.0/3.0 da como resultado 0.66666

Si hay operandos de diferentes tipos de datos, se convierten al tipo de datos más amplio y el tipo del valor resultante es del tipo

más amplio. Por ejemplo, si hay enteros y flotantes, todos los números se convierten a flotantes y el resultado se calcula como

flotante.

Por ejemplo:

4/3.0 da como resultado 1.3333

El operador **%** calcula el residuo de la división entera y sólo existe para datos de tipo entero

Por ejemplo:

10%3 da como resultado 1

Otros operadores de Asignación

En C++ es posible abreviar algunas expresiones de asignación como se muestra en la siguiente tabla:

Operador	Expresión equivalente
v + = e	v = v + e
v - = e	v = v - e
v * = e	v = v * e
v / = e	v = v / e
v % = e	v = v % e

Otros Operadores aritméticos

En C++ existen también los siguientes operadores aritméticos:

++ incremento

-- decremento

Es decir:

x++ ó ++x es equivalente a x = x+1

x-- ó --x es equivalente a x = x-1

Estos operadores son unitarios, es decir, trabajan con un solo operando y solamente se pueden utilizar con variables de tipo

entero.

Los operadores se pueden utilizar antes o después del nombre de la variable y funcionan de diferente manera:

- Si se ponen antes, primero se realiza la operación (incremento o decremento) y luego se utiliza el valor de la variable en la

  expresión en la que se encuentre.

- Si se pone después, primero se utiliza el valor de la variable en la expresión y luego se lleva a cabo la operación (incremento

  o decremento).

Por ejemplo:

Supón que a = 10 y c = 4

La operación v = a * c++; v toma el valor de 40 y c queda con el valor de 5

La operación v = a * ++c; v toma el valor de 50 y c queda con el valor de 5

Jerarquía de los operadores aritméticos

Prioridad	Operadores	Asociatividad
1	( )	Empezando por los paréntesis más internos
2	++, --, +(positivo), -(negativo)	De derecha a izquierda, ++ y -- dependiendo de la posición
3	*,/,%	De izquierda a derecha
4	+, -	De izquierda a derecha
5	=,+=,-=,*=, /=,%=	De derecha a izquierda

Algunas Funciones Matemáticas Predefinidas

C++ contiene una serie de funciones matemáticas que puedes utilizar en tus programas, algunas de estas funciones son:

abs(x) obtiene el valor absoluto de x, x debe ser entero

sqrt(x) obtiene la raíz cuadrada de x, x debe ser positivo

pow(x,y) calcula x elevado a la potencia y, pueden ser enteros o flotantes

el uso de estas funciones requiere de la librería <math.h>

Operadores de Relacionales

Los operadores relacionales que tiene C++ son :

Operador en C++	Significado
==	Igual
!=	Diferente
<	Menor que
>	Mayor que
<=	Menor o igual que

>=	Mayor o igual que

Operadores Lógicos

Los operadores lógicos que maneja C++ son:

Operador en C++	Significado
||	or
&&	and
!	not

El resultado de las expresiones que incluyen operadores relacionales o lógicos generan resultados verdaderos o falsos. Para C++ el

cero representa falso y cualquier otro número verdadero.

Ejemplos de construcción de expresiones

Expresión para saber si un número es par:

(num % 2 == 0)

Expresión para saber si un número A está en el rango 5 a 300 incluyendo los extremos

(num > = 5) && (num <=300)

Ejercicios

I. Realiza las siguientes expresiones en C++ para obtener el valor con el que quedará la variable x.

1. x = 7 + 3 * 6 / 2 - 1; ver solución

2. x = (3 * 9 * (3 + (9 * 3 / (3)))) ; ver solución

Dados los valores iniciales de a = 15, b = 3, cual será el valor final de las variables después de ejecutar las siguientes expresiones (individualmente).

3. a = a * 6 / a ++ ; ver solución

4. b = --b * a++ / b; ver solución

Construye las siguientes expresiones.

5. Expresión para saber si 3 número son iguales.

6. Expresión para determinar que la variable edad está fuera del rango de 15 a 40 sin incluir los extremos

ver solución de ambos ejercicios

Material de Apoyo

Presentacion de Expresiones

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Estatuto if

Sintaxis

```
if ( condición

        estatuto;

 else                                   // la parte else es opcional

        estatuto;
```

En caso de requerir más de un estatuto es necesario usar llaves.

```
if ( condición )
 {
        bloque de estatutos;
 }

 else                                   // la parte else es opcional
{
        bloque de estatutos;
 }
```

Ejecución del if

La estructura if (sin else), ejecuta el estatuto solo cuando la condición es verdadera; en caso de que sea falsa brinca el estatuto

(o estatutos en caso de tener más de uno entre llaves)

La estructura de selección if / else , ejecuta la(s) accion(es) después de la parte if cuando la condición es verdadera; en caso de

que sea falsa ejecuta la(s) acción(es) que está(n) después del else.

Ejemplo

```
if (promedio >= 70)
     cout << " Aprobado";
else
     cout << "Reprobado";
```

if´s anidados

Se dice que hay if anidados cuando existe un if/else dentro de otra estructura if/else

Ejemplo:

Determinar si un número es positivo, cero o negativo.

```
      if (num > 0)
       cout << " El número es positivo";
   else
       if (num == 0)
           cout << " El número es cero";
       else
           cout << " El número es negativo";
```

Si tenemos el siguiente fragmento de código:

```
      if (condición 1)
          if (condición 2)
               estatuto;
      else               // este else pertenece al if de la condición 2, pues
se asocia al if más cercano
           estatuto;
```

Si queremos que el else pertenezca al primer if debemos poner:

```
      if (condición 1)
      {
         if (condición 2)
               estatuto;
      }
      else               // con el uso de llaves cerramos el if anidado y el
else
         estatuto;       // pertenece al primer if
```

Ejemplos de Programas

A continuación tendrás algunos ejemplos de programas en C++:

Ejemplo I: Programa que calcula la energía necesaria para la combustión de un compuesto dada la cantidad de átomos x e y de dos diferentes elementos, de acuerdo a la siguiente fórmula:

$$4x + 3x^2y - 2y \text{ si } x > 0 \text{ , } y >= 0$$

$e(x,y) = x^2 - 4(y - x) + y^2$ si $x < 0$, $y >= 0$

$x^2 + y + y (x- 2)$ en cualquier otro caso

```
#include <iostream.h>
#include <math.h>    // requerimos esta librería para elevar a una potencia con
pow

/* Declaración de variables globales */

      double x, y, res;           // declaramos las variables como dobles

 int main()
 {
   cout << "Dame los valores de x e y "<< endl;
   cin >> x >> y;
   if (y >= 0)
      if ( x > 0)
           res = 4 * x + 3 * pow (x,2) * y - 2 * y;
      else
           res = pow(x,2) - 4* (y - x) + pow (y,2);
   else
      res = pow (x,2) + y + y * (x-2);
   cout << "El resultado es " << res << endl;
   return 0;
   }
```

Ejemplo II: Programa que lee 3 números enteros diferentes y los despliega de mayor a menor.

```
#include <iostream.h>

/* Declaración de variables globales */

      int a,b,c;

int main()
{
    cout << "Dame los 3 numeros "<< endl;
    cin >> a >> b >> c;
    if (a > b)
       if ( b > c)
           cout << a << " " << b << " " << c;
       else
           cout << a << " " << c << " " << b;
   else
       if ( a > c)
           cout << b << " " << a << " " << c;
       else
           if ( b > c )
               cout << b << " " << c << " " << a;
           else
               cout << c << " " << b << " " << a;
    return 0;
}
```

Estatuto Switch

Se utiliza para ejecutar acciones diferentes según el valor de una expresión o variable. Consiste en una serie de de etiquetas case

y un case por omisión (default)

Sintaxis

```
switch ( variable)     la variable es de tipo entero o caracter
{
    case valor1 : accion1; break;
    case valor2 : accion2; break;
        .
        .
    case valor n : accionn;break;
    default: accionD;
 } ;
```

La acción 1 se ejecuta si la variable adquiere el valor1.

La acción 2 se ejecuta si la variable adquiere el valor 2.

La acción n se ejecuta si la variable adquiere el valor n.

Cualquier otro valor de la variable conduce a la realización de la secuencia accionD, indicada por la palabra reservada default.

La secuencia default puede omitirse

Estatuto break

Cuando se encuentra una sentencia `case` que concuerda con el valor del `switch` se ejecutan las sentencias que le siguen y todas las demás a partir de ahí, a no ser que se introduzca una sentencia `break` para salir de la sentencia `switch`.

Ejemplos:

Ejemplo I: Programa que pide un número de mes y escribe la cantidad de días que tiene.

```
#include <iostream.h>

int mes;

void main()
{
  cout<<"Teclea el mes"<<endl;
  cin>>mes;
  switch (mes)
  {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: cout<<"31 dias"<<endl; // el break va hasta después del 12 pues si fue
cualquiera de estos meses tiene 31
                              break;     // días, y al encontrar el break sale del
```

```
switch, si no fue ninguno sigue revisando
        case 2:  cout<<"28 o 29 dias"<<endl;
                                break;
        case 4:
        case 6:
        case 9:
        case 11: cout<<"30 dias"<<endl; break;
        default: cout<<"ERROR";
       };
     }
```

Ejemplo II : Dada una fecha con números, muestra la fecha poniendo el mes con letras.

```
#include <iostream.h>

int dia, mes, anio;
char resp;

void main()
{
       cout<<"Teclea el dia"<<endl;
       cin>>dia;
       cout<<"Teclea el mes"<<endl;
       cin>>mes;
       cout<<"Teclea el anio"<<endl;
       cin>>anio;
       cout<<dia<<" de ";
       switch (mes)
       {
              case 1: cout<<"Enero"; break;
              case 2: cout<<"Febrero"; break;
              case 3: cout<<"Marzo"; break;
              case 4: cout<<"Abril"; break;
              case 5: cout<<"Mayo"; break;
              case 6: cout<<"Junio"; break;
              case 7: cout<<"Julio"; break;
              case 8: cout<<"Agosto"; break;
              case 9: cout<<"Septiembre"; break;
              case 10: cout<<"Octubre"; break;
              case 11: cout<<"Noviembre"; break;
              case 12: cout<<"Diciembre"; break;
              default: cout<<"ERROR";
       };
       cout<<" de "<<anio<<endl<<endl;
}
```

Ejercicio

Resuelve los siguientes problemas en C++

1.Dadas las coordenadas (x,y) del plano cartesiano desplegar el cuadrante en el que se encuentra. ver solución

2. Escribe un programa que pida al usuario 2 números enteros que representan una carta, el

primer número representa el palo (1-diamantes, 2- tréboles, 3-corazones, 4-espadas) y el segundo representa el número de la carta (1-As, 2, 3, ... 10, 11-J, 12-Q, 13-K). El programa debe mostrar en la pantalla el nombre de la carta; en caso de que alguno de los números no sea válido debe decir que la carta no existe. Por ejemplo:

Si los números son 3, 1	el programa mostrará	As de Corazones	
Si los números son 2, 8	el programa mostrará	8 de Trévoles	
Si los números son 4, 12	el programa mostrará	Q de Espadas	
Si los números son 1, 16	el programa mostrará	La carta no existe	ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Módulo 2: Estatutos de Control

2. Ciclos

Estatuto while

Sintaxis

while (condición)

estatuto;

Si se requiere realizar más de un estatuto se deben utilizar llaves.

while (condición)

{

bloque de estatutos;

}

Aquí se ejecuta el (los) estatuto (s) mientras la condición es verdadera; al momento de ser falsa termina el ciclo.

Si la condición es falsa la primera vez nunca se ejecuta(n) el (los) estatuto(s).

Ejemplos:

Ejemplo I: Programa que pide 10 números y calcula y muestra la suma de todos los que fueron pares.

```
#include <iostream.h>

int suma=0, cont= 10, num;

int main()
{
  while (cont > 0)
 {
    cout << "Dame el numero " << endl;
    cin >> num;
    if (num % 2 == 0 )       // verificamos que sea un número par
       suma += num;          // recordemos que suma+= num es equivalente a suma =
suma + num (ver modulo1 sección 3)
    cont--;
  }
  cout << "La suma de los numeros pares proporcionados es " << suma << endl;
  return 0;
}
```

Ejemplo II : Obtener el promedio de calificaciones de un grupo de 20 estudiantes

```
#include <iostream.h>

int total, contador, calif;
double promedio;

int main()
{
       total = 0;
       contador = 1;
```

Ejercicio

Realiza los siguientes programas:

1. Una compañía paga $5,000 semanales a sus vendedores más una comisión de 9% del total de ventas realizadas. Realiza un programa que determine el pago que deberá realizarse a cada vendedor. Una vez que ha calculado el pago de cada vendedor deberá mostrar la cantidad de vendedores cuyo pagó excedió los $8,000. ver solución

2. Escribe un programa que tome un número entero y obtenga la suma de sus dígitos pares y la suma de sus dígitos nones.
ver solución

3. Escribir un programa que obtenga la suma de n términos determinada por la serie:

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Ciclos anidados

Se dice que un programa tiene ciclo anidado cuando un el estatuto de un ciclo (while, do..while o for) es a su vez otro ciclo.

Ejemplo: Obtener el resultado de la serie $\sum_{i=1}^{n} \sum_{j=2}^{m} i*j$

```
#include <iostream.h>

 int n,m,resultado= 0;

 int main()
 {
  cout << "Dame el valor de n y m " << endl;
  cin >> n >> m;
  for ( int i =1 ; i <= n; i++ )
     for ( int j =2 ; j <= m; j++ )
          resultado = resultado + i*j;
  cout << " El resultado de la serie es " << resultado;
  return 0;
 }
```

Ejemplo : Programa que dado un número entero size, construye una pirámide de asteriscos de altura size.
Por ejemplo, si size = 5, el programa mostrará:

```
*
**
***
****
*****
```

```
#include <iostream.h>

int i,j,size;

int main()
{
   cout << "Dame el tamaño "<<endl;
   cin >> size;
   for (i=1;i<=size;i++)
   {
      for (j=1;j<=i;j++)
          cout << "*";
      cout<<endl; }              // para brincar a la siguiente línea
```

```
   return 0;
}
```

Ejemplo : Mismo problema pero ahora construye la pirámide en otro orden
Por ejemplo, si size = 5, el programa mostrará:

```
*****
 ****
  ***
   **
    *
```

```
#include <iostream.h>

int i,j,k,size;

int main()
{
   cout << "Dame el tamaño "<<endl;
   cin >> size;
   for (i=size;i>=1;i--)
   {
      for (k=i;k<size ;k++)    //para dejar los espacios en blanco
            cout << " ";
      for (j=1;j<=i;j++)
            cout << "*";
      cout<<endl;

    }
return 0;
}
```

Ejemplo: Se desea que dados dos números enteros n1 y n2 se obtenga la multiplicación de n1 por n2 usando el algoritmo de la multiplicación rusa. El algoritmo dice lo siguiente: realiza la división entera de n1 entre 2 y al mismo tiempo multiplica n2 por 2 repetidamente hasta que n1 sea 1. Sume todos los valores de n2 que corresponden a los valores de n1 impares y ese es el valor de la multiplicación.

```
#include <iostream.h>

int n,m,suma=0;

int main()
{
   cout << "Dame los numeros a multiplicar "<<endl;
   cin >> n>>m;
   do
   {
      if (n%2 !=0)   // n es impar
          suma+=m;   // sumo el valor de m
      n=n/2;
      m=m*2;
   }
   while (n > 1);
   if (n%2 != 0)
       suma+=m;
   cout << "El resultado es "<<suma<<endl;
   return 0;
}
```

Ejercicio

Realiza los siguientes programas.

1. Dado un número entero N, se desea que muestre en la pantalla la serie formada por todos los números nones que van desde 1 hasta N (si N fuera par entonces la serie termina en N-1) en la serie el primer término (el 1) tiene signo positivo y luego el signo se va alternando en los siguientes términos. Después de mostrar la serie, debe también mostrar el resultado de la suma, como se muestra en los ejemplos.

Por ejemplo, si el usuario teclea el número 9 la serie que se mostrará en la pantalla es la siguiente:
1 - 3 + 5 - 7 + 9 = 5

Otro ejemplo, si el usuario teclea el número 12, la serie que se mostrará en la pantalla es:
1 - 3 + 5 - 7 + 9 - 11 = - 6 ver solución

2. Leer un entero positivo N y averiguar si es perfecto. Un número es perfecto cuando es igual a la suma de sus divisores excepto el mismo.

Ejemplo:
Si N = 6 sus divisores son 1, 2 y 3 si los sumamos nos da 6 por lo tanto es perfecto.
ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

1. Declaración y Definición de Funciones

En C++ todas las acciones se llevan a cabo mediante funciones. En C++ el programa también es una función llamada main.

-

Declaración de una función

tipo nombre (lista de parámetros)

```
{
        declaración de variables locales;

         estatutos;

         return valor;

 }
```

donde:

tipo es el valor de retorno de la función. Si la función no regresa ningún valor se pone el tipo void para indicarlo.

nombre es un identificador, el cual se recomienda sea un nombre representativo de lo que hace la función.

lista de parámetros es una lista de cero, una o más declaraciones de variables (parámetros) separadas por coma. Cada

parámetro se debe de declarar con su tipo de dato. Este tema se explicará a detalle en la sección 3 del módulo.

Notas sobre las funciones en C++:

- En C++ las funciones pueden declararse en 2 lugares distintos:

1. Después de la declaración de variables globales y antes de la función main() o la función que la vaya a llamar.

2. Después de la función main() o de la función que la vaya a llamar, en cuyo caso deben ser identificadas antes, para

ello debe declararse solamente el encabezado de la función, al que se llama **prototipo**, después de la declaración de variables

y antes de la función main(). Esta función prototipo le indica al compilador el tipo de dato que regresará la función, el

número de parámetros que la función espera recibir, el tipo de estos parámetros y el orden en el cual los parámetros

deben ser esperados. El compilador usa esta función prototipo par validar las llamadas a las funciones. En algunas

versiones de C no se realiza este tipo de verificación, pero en el caso de C++ si.

- Una función no puede declararse dentro de otra función.
- Toda función que no es void debe tener un return.
- Al llegar al estatuto return, la función se deja de ejecutar y envía el valor especificado como valor de retorno.

Orden de ejecución de las funciones en C++:

Como ya dijimos C++ se basa en funciones siendo main() la función principal. por lo que:

- La ejecución de un programa empieza en la función main().
- Las otras funciones se ejecutan cuando se mandan llamar.
- Si una función no se manda llamar, nunca se ejecuta.
- Como se comentó en las notas anteriores, para que una función sea llamada tiene que declararse antes.

Llamada a una función

Hay varias formas de llamar a una función, dependiendo de lo que se desea hacer con el resultado o resultados que regrese, pero la

regla que siempre se conserva es que la lista de parámetros debe coincidir con los valores que recibe la función en su declaración

en tipo y cantidad.

- Un estatuto de asignación si el valor que regresa es necesario en otra operación

  a = **nombre** (lista de parámetros);

- Una expresión

  a = cantidad ***** **nombre** (lista de parámetros);

- El estatuto de salida, si solo nos interesa imprimir el resultado, en caso de que no haya parámetros de referencia

  cout << **nombre** (lista de parámetros);

- Si hay parámetros de referencia, debemos llamar a la función y luego imprimir los valores de los argumentos que hayan sido

  modificados

  nombre (lista de parámetros);
  cout << parámetros;

- Una condición

  if (**nombre** (lista de parámetros) > 7)

  while (**nombre** (lista de parámetros) > 7)

Ejemplos

Ejemplo 1: Se necesita una función que calcule la distancia entre 2 puntos con coordenadas (Xa, Ya) y (Xb, Yb) . Para calcularla se usa la fórmula:

DISTANCIA =

```
#include <iostream.h>
#include <math.h>      //librería para usar las operaciones pow y sqrt

double distancia (double xx1, double yy1, double xx2, double yy2)
{
   double result,p1,p2;
   p1 = pow((xx2-xx1),2);
   p2 = pow((yy2-yy1),2);
   result = sqrt(p1+p2);
   return result;
}

int main()
{
   double x1,y1,x2,y2;
   cout << "Dame las coordenadas del 1er punto "<<endl;
   cin >> x1>>y1;
   cout << "Dame las coordenadas del 2do punto "<<endl;
   cin >> x2>>y2;
```

```
   cout<<"La distancia entre los 2 puntos es = "<<distancia(x1,y1,x2,y2)<<endl;
   return 0;
}
```

Ejemplo 2: Se desea obtener el perímetro de un triángulo dadas las coordenadas de los 3 puntos.

Usaremos la función de distancia del problema anterior para resolverlo

```
#include <iostream.h>
#include <math.h>      //librería para usar las operaciones pow y sqrt

double distancia (double xx1, double yy1, double xx2, double yy2)
{
   double result,p1,p2;
   p1 = pow((xx2-xx1),2);
   p2 = pow((yy2-yy1),2);
   result = sqrt(p1+p2);
   return result;
}

int main()
{
   double x1,y1,x2,y2,x3,y3,p12,p13,p23;
   cout << "Dame las coordenadas del 1er punto "<<endl;
   cin >> x1>>y1;
   cout << "Dame las coordenadas del 2do punto "<<endl;
   cin >> x2>>y2;
   cout << "Dame las coordenadas del 3er punto "<<endl;
   cin >> x3>>y3;
   // calculando la distancia entre el punto 1 y 2
   p12 =distancia(x1,y1,x2,y2);
   // calculando la distancia entre el punto 1 y 3
   p13 =distancia(x1,y1,x3,y3);
   // calculando la distancia entre el punto 2 y 3
   p23 =distancia(x2,y2,x3,y3);
   cout << "El perimetro es " <<p12+p13+p23<<endl;
   return 0;
}
```

Ejercicio

Identifica si son correctas las siguientes llamadas a una función llamada Compara cuyo encabezado es el siguiente:

double Compara (int x, double y);

1. int a,b;
   double z;

   cout << Compara (a,b,z);

2. double a,b;

```
   int  z;

   if Compara(a,z)


3. double a,b,c;
  do
  {
     .....
   }
   while (compara
(a,b));
```

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

2. Variables locales, globales y alcance de las variables

Variables globales

Son las variables declaradas después de la declaración de librerías y antes de la declaración de funciones. Estas variables son

reconocidas en cualquier función del programa.

Variables locales

Todas las variables declaradas en una función son variables locales, estas variables son reconocidas en la función en la cual fueron

definidas. Los parámetros de una función son también variables locales de ésta.

Un error común es declarar con el mismo nombre una variable local y un parámetro.

La declaración de una variable local en una función debe realizarse antes de ser usada, lo más usado es declarar las variables

después del encabezado de la función y antes de las instrucciones, ejemplo:

```
int calcula (int a, int b)
{
     double x;
     x = a *b++;
}
```

```
Pero también pude declararse entre las instrucciones antes de ser usada,
ejemplo:
```

```
int ejemplo (double num)
{
   num++;
   double valor;
   valor = 5 * num;
}
```

Variables declaradas dentro de un ciclo for

En un ciclo for en la sección de inicialización puede declararse la variable del ciclo, ejemplo

```
for (int x=1; x<=3; x++)
```

en estos casos la variable x solo es reconocida en el ciclo for y al salir de ahí aún cuando esté

en una función no es reconocida y

marcará un error de sintaxis si se hace uso de ella.

Es importante notar que si tenemos la siguiente declaración en un for

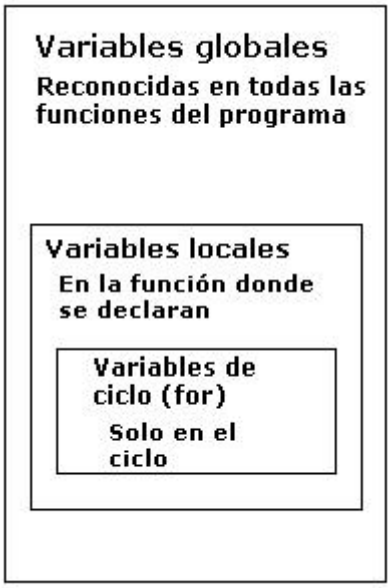
```
for (int z=1, y=2; z!=5; z++)
```

tanto la variable z como y son reconocidas solo en el for.

Alcance de las variables

Podemos resumir en forma gráfica lo explicado anteriormente, donde depende en qué lugar se haya declarado una variable es el

alcance (lugares donde es reconocida y puede usarse).



Las variables que son declaradas como globales no deben usarse como parámetros en las funciones, dado que son reconocidas en

todas las funciones y esto sería un error de programación, no de sintaxis ni lógica.

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

3. Parámetros

Los parámetros son la forma de comunicar valores, que se han almacenado en variables locales, entre las funciones de un programa.

Declaración de los parámetros en la definición de la función

tipo nombre (lista de parámetros)

```
{
        declaración de variables locales;

         estatutos;

         return valor;

}
```

Lista de parámetros es lista de cero, una o más declaraciones de variables separadas por coma. Cada parámetro debe de tener su

tipo antes del nombre. Un error común es cuando se tiene más de un parámetro del mismo tipo solo se le pone tipo al primero y se

separa por comas.

Ejemplo: Supongamos que queremos declarar una función que regresará un valor doble y recibe dos valores enteros num1 y num2.

La declaración correcta es: double ejemplo (int num1, int num2)

El típico error es poner: double ejemplo (int num1, num2) ó no poner tipo double ejemplo (num1, num2)

Tipos de parámetros en C++

1. Por valor

2. Por referencia

Parámetros por valor

Cuando un argumento es pasado por valor, se hace una copia del valor del argumento y se

pasa a la función que es llamada. Los

cambios a la copia no afectan el valor original de la variable que aparece en la llamada. Una desventaja del paso por valor es que una gran

cantidad de datos es pasado y copiado puede tomar un cantidad considerable de tiempo de ejecución y espacio de memoria. A

este tipo de parámetros se les conoce como parámetros de *"entrada"*

Ejemplos

Ejemplo 1: Programa que calcula el área de un círculo

```
#include <iostream>
#include <math>
using namespace std; //usando el namespace


double r;

double calc_area (double rad)   // rad es parámetro por valor recibe el valor
de r
{ double area;                  // declaracion de la variable local
  area = 3.14*pow(rad,2);
  return area;                  // se regresa a la función main() el resultado
}

int main()
{
cout << "Dame el radio "<<endl,
cin >> r;
cout << "El area es del circulo es "<<calc_area(r)<<" con un radio "<< r<<endl;
return 0;                     // el valor de r no cambia al efectuarse la
llamada a la función calc_area
}
```

Ejemplo 2: Desarrolla un programa que tenga la función factorial para calcular la aproximación de e (número de Euler).
La aproximación debe basarse en la siguiente fórmula:

$$1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots\ldots$$

```
#include <iostream.h>

int fact(int n)
{  int pot=1;
   for (int i=1;i<=n;i++)
      pot = pot *i;
    return pot;
}

void main()
{
   int euler;
   double suma=1;
   cout << "Dame la aproximacion que quieres "<<endl;
   cin >> euler;
   for (int i=1;i<=euler;i++)
      suma = suma + 1.0/fact(i);
```

```
   cout << "El valor de la aproximacion es "<< suma<<endl;
}
```

Notas importantes:

- Para evitar ambigüedad, se recomienda no usar el mismo nombre en los argumentos pasados a una función y los

  correspondientes parámetros en la definición de la función.

- En la llamada a una función el tipo y número de parámetros debe coincidir con los de la definición de la función.

Referencias y parámetros de referencia

Un parámetro por referencia es un alias para su argumento correspondiente en la llamada a la función. Para indicar que un

parámetro es pasado por referencia, se pone un & después del tipo de dato y antes del nombre de la variable. Estos parámetros se

nombran comúnmente como parámetros de "entrada/salida", pues estos alteran el valor el valor del argumento al momento de

acabar la llamada a la función.

Este tipo de parámetros es usado cuando se requiere una función que regrese más de un valor.

Ejemplos

Ejemplo 1: Función llamada Separa que reciba como parámetro un número X de tipo double, y que regrese dos parámetros de tipo

double: en el primero de ellos deberá regresar la parte entera del número X y en el segundo la parte fraccionaria del número X.

Por ejemplo:

al llamar Separa (32.45, a, b);

la variable a tomará el valor de 32.0 y la variable b tomará el valor de 0.45.

```
#include <iostream.h>

int Uno (double x1, int &a1, double &b1)
{
  a1 = x1; // al pasarlo a un entero pierde la parte fraccionaria
```

```
  b1=(x1-a1)*100;
  return 0;
}

int main()
{
 double x,b;
 int a;
 x=45.12;
 Uno(x,a,b);   // Llamada a la función con dos argumentos que cambiarán de valor
 cout<<"Parte entera = "<<a<<endl;
 cout<<"Parte fraccionaria = "<<b<<endl;
 return 0;
}
```

Ejemplo 2: Programa que lea un entero n que representa la cantidad de números a leer y la mande a una función que deberá regresar el valor más grande y más chico de los números leídos.

```
int bigsmall(int n, int &mayor, int &menor)
{   int num;
   // vamos a leer el primero asumiendo que es tanto mayor como menor
   // para después comparar con los leídos
   cout <<"Dame el numero "<<endl;
   cin >> num;
   mayor = num;
   menor = num;
   for (int i=2;i<=n;i++)
   {
      cout <<"Dame el numero "<<endl;
      cin >> num;
      if (num > mayor)
          mayor = num;
      if (num < menor)
          menor = num;
   }
   return 0;
}

void main()
{
   int numer,may,men;
   cout << "Dame la cantidad de numeros a leer "<<endl;
   cin >> numer;
   bigsmall(numer,may,men);
   cout << "El valor mayor es "<< may<<endl;
   cout << "El valor menor es "<< men<<endl;
}
```

Ejercicio

Escribe una función que reciba como parámetro la fecha de nacimiento de una persona y la fecha actual y que calcule la edad de la persona, si tiene menos de un año que dé la edad en meses.

Por ejemplo:

Si la fecha de nacimiento es 12 de Oct de 2000 y la fecha actual es 1 de Oct de 2003 el programa debe decir que la persona tiene 2 años cumplidos.

Si la fecha de nacimiento es 15 de Marzo de 1980 y la fecha actual es 29 de Marzo de 2003, el programa debe decir que la persona tiene 13 años.

Si la fecha de nacimiento es el 12 de Noviembre de 2002 y la fecha actual es el 15 de Agosto de 2003 el programa debe decir que la persona tiene 9 meses de edad.

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Módulo 4: Clases y Objetos

1. Programación Orientada a Objetos

Conceptos de Programación Orientada a Objetos

Empecemos con un ejemplo:

Supón que estas en un estacionamiento caminando hacia tu auto, entonces tomas el control remoto y

oprimes un botón, en ese momento se desactiva la alarma de tu automóvil.

El control remoto es un objeto físico que tiene propiedades: peso, tamaño y también puede hacer cosas:

puede enviar mensajes al automóvil. No está claro como hace esto, pero no necesitas saberlo para poder usarlo; solo necesitas

saber qué botón oprimir para que esto ocurra.

Los botones son la **interface** del control remoto. Si conoces la interface de un objeto, lo puedes usar para realizar acciones sin

entender cómo trabaja.

El automóvil también es un objeto físico, tiene propiedades y realiza acciones, una de ellas es recibir un mensaje de un control

remoto para desactivar la alarma.

Gracias a la **abstracción** eres capáz de hacer que dos objetos interactúen entre sí, sin necesidad de entender cómo funcionan.

Esto también permite que uses otros controles remotos y otros automóviles de otras marcas y otros modelos.

La clave para desarrollar un sistema utilizando los conceptos de Orientación a Objetos es definir qué objetos forman parte del

sistema, crear abstracciones apropiadas para ellos y separar la implementación interna de su comportamiento externo.

Desde el punto de vista de la programación orientada a objetos, la abstracción consiste en ocultar los detalles irrelevantes para

que el objeto pueda ser comprendido y utilizado por el usuario. Por un lado, esto disminuye la complejidad permitiendo que el

usuario utilice los datos del objeto sin necesidad de conocer detalles, y por otro lado protege los datos evitando que el usuario

pueda llegar a ellos y modificarlos sin utilizar las funciones diseñadas para ello, afectando con esto la confiabilidad del sistema.

Pero, ¿Qué es un objeto?

Un **objeto** puede ser alguna cosa física, como una pluma, un escritorio, un gato; pero también hay cosas abstractas que son

objetos, como un número o una cuenta bancaria.

Sabemos que algo es un objeto si tiene nombre, se puede describir en base a sus características o propiedades y es capaz de

hacer algo o comportarse de cierta manera.

Por ejemplo:

Pensemos en una cuenta bancaria sencilla, podemos describirla como un objeto de la siguiente manera:

- Clase: Cuenta Bancaria
- Propiedades: Nombre del dueño, saldo actual, tipo de cuenta.
- Comportamientos: depositar, retirar, consultar el saldo.

Los lenguajes orientados a objetos nos permiten crear abstracciones encapsulando las propiedades y los comportamientos en un

solo concepto llamado **clase**. Las propiedades (también llamadas atributos o datos miembro) que pertenecen a una clase se

definen como variables y los comportamientos (también llamados métodos o funciones miembro) se definen como funciones, y

éstos tienen la característica de tener acceso a los atributos privados de la clase.

La diferencia entre clase y objeto es sutil pero muy importante. La clase es el concepto abstracto, es una especie de molde para

crear objetos, define qué atributos y métodos tiene la clase. Por otro lado el objeto es la entidad concreta. A partir de una clase,

se pueden crear instancias, es decir objetos y cada objeto tiene valores distintos para sus atributos.

Por ejemplo:

Continuando con nuestro ejemplo de la clase cuenta bancaria,

La clase es Cuenta Bancaria.

Los objetos podrían ser

Cuenta 193	Cuenta 281
Nombre del dueño: Jorge López	Nombre del dueño: Laura Treviño
saldo actual: 5,248.00	saldo actual: 3,915.00
tipo de cuenta: ahorros	tipo de cuenta: chequera

Un objeto puede recibir **mensajes**, los cuales son para para pedirle que haga algo, o bien, que cambie alguna de sus propiedades.

Esto es equivalente a llamar a una función, solo que al llamar al método se tiene que hacer referencia al objeto específico en el

que se ejecutará dicho método.

Por ejemplo, se puede pedir a la cuenta bancaria 193 ejecutar el método depositar con el valor 100, lo cual se reflejará con un

cambio en el valor del atributo saldo actual, el nuevo valor para dicho atributo será 5,348.00. Pero nota que el valor del atributo

saldo actual de la cuenta 281 permanece sin cambio.

Funciones para Establecer y Consultar el valor de los atributos

Es muy común que las clases incluyan métodos para establecer (o escribir) y consultar (o leer) los valores de atributos privados de

la clase; a estos métodos se les llama en inglés "mutators" y "accesors" y se acostumbra poner los nombres "set" seguido del

nombre de la variable para métodos "mutators" y "get" seguido del nombre de la variable para los métodos "accesors"; así por

ejemplo deberíamos tener los siguientes métodos:

```
void setSaldoActual(double valor);     // método para establecer el valor de la variable saldo
Actual.
double getSaldoActual();               // método para consultar el valor de la variable saldo
Actual.
```

Tener estos métodos puede parecer equivalente a haber declarado el atributo como público;

sin embargo hay una pequeña

diferencia que es muy imortante. Si hacemos el atributo público cualquier programa cliente puede modificar su valor sin verificar

que el nuevo valor que se le asigne sea un valor apropiado; por otro lado, un método para establecer el valor de la variable puede

realizar la validación del dato que va a asignar al atributo y si no es un dato válido puede rechazarlo.

Ejercicio

A partir del siguiente texto, identifica los atributos y métodos para la clase Rectángulo.

En un paquete de dibujo uno de los objetos que permite dibujar en la pantalla es un rectángulo, en este paquete se puede

establecer el tamaño del rectángulo con el movimiento del mouse, se puede establecer el grosor de las líneas del rectángulo, así

como su color, se puede establecer si tiene o no un color de relleno y escoger dicho color y se puede posicionar en cualquier

coordenada de la pantalla. ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Cómo se define una clase en C++

Una clase puede contener partes públicas y privadas. Por default, los elementos que son definidos dentro de una clase son

privados, esto quiere decir que no pueden ser accesados desde afuera de la clase, también se puede especificar cuales elementos

serán privados con la palabra private.

Para hacer algunos elementos de la clase públicos se utiliza la palabra public, los elementos que se definen después de esta

palabra son públicos, lo que significa que pueden ser utilizados desde afuera de la clase.

Solo los métodos de la clase pueden accesar los atributos privados de la misma.

El formato para declarar una clase es el siguiente:

```
class NombreClase
{
   public:
      // declaración de atributos y métodos públicos

   private:
      // declaración de atributos y métodos privados
};
```

Ejemplos

Ejemplo 1 Supongamos que requerimos declarar la clase cícrulo para un programa en el que se requiere calcular el área del

círculo, podemos declarar la clase como se muestra en el siguiente ejemplo:

```
class Circulo
{
   public:
        double CalcArea( );
        void setRadio(double radio);
        double getRadio();
   private:
        double radio;
};
```

Nota que en esta declaración las funciones no se declaran completas, solamente se declara su prototipo.

Las funciones se definen después, utilizando un formato en el que se tiene que especificar de que clase es miembro este método,

como se muestra en el siguiente formato:

```
tipo NombreClase :: NombreMétodo ( lista de parámetros)
{
     estatutos;
}
```

continuando con el ejemplo:

```
double Circulo::CalcArea( )
{
     return 3.1416*radio*radio;
}

void Circulo::setRadio( double ra)
{
     radio = ra;
}

double Circulo::getRadio()
{
     return radio;
}
```

Dentro de un programa de C++ una instancia u objeto de la clase se representa como una variable, por lo que para crear una

instancia declara una variable cuyo tipo de dato es el nombre del objeto, se usa el siguiente formato:

NombreClase NombreVariable;

para nuestro ejemplo:

Circulo A;

en este caso estamos declarando la variable A que es un objeto de tipo Circulo; nota que el nombre de la clase es el tipo de dato.

Un mensaje pide a un objeto que ejecute alguno de sus métodos. Para enviar un mensaje a un objeto se usa el siguiente formato:

NombreObjeto.NombreMétodo (parámetros);

para nuestro ejemplo:

A.setRadio(10);

en este caso estamos llamando al método setRadio de la clase Circulo.

Ejemplo 2 El siguiente ejemplo muestra como se puede crear y usar un objeto dentro de una aplicación:

```
int main( )
{
   Circulo A;     // se crea el objeto A de la clase Circulo
   double area;

   A. setRadio(10);   // se envía el mensaje al objeto A para que
ejecute el método setRadio
   area = A.CalcArea( );   // se envía el mensaje al objeto A para
que ejecute el método CalcArea

   return 0;
}
```

Es común colocar la declaración y definición de una clase en un archivo y poner a ese archivo el nombre de la clase con extensión

.h. Para nuestro ejemplo podemos utilizar:

Circulo.h

Y para usar la clase Circulo en un programa cliente, se incluye la clase con un #include. Por ejemplo:

#include ?Circulo.h?

El ejemplo anterior completo queda como sigue:

```
// Esto se encuentra en el archivo Circulo.h
class Circulo
{ public:
     double CalcArea( );
     void setRadio(double ra);
     double getRadio();
private:
    double radio;
};


double Circulo::CalcArea( )
{
    return 3.1416*radio*radio;
}


void Circulo::setRadio( double ra)
{
     radio= ra;
}


double Circulo::getRadio()
{
     return radio;
}


// Esto se encuentra en la aplicación UsaCirculo.cpp
#include <iostream.h>
#include "Circulo.h"
```

```
int main( )
{
    Circulo A;
    double area;

    A. setRadio(10);
    area = A.CalcArea( );
    cout<<area;

    return 0;
}
```

Ejercicio

Copia los ejemplos que se incluyen en el material y pruébalos.

Define una clase Rectángulo que tenga las siguientes especificaciones:

Atributos:

- largo
- ancho

Métodos:

- calcula Area
- setLargo
- setAncho
- getLargo
- getAncho

Utiliza esta clase para resolver el siguiente problema:

Hay una pared rectangular que tiene una ventana rectangular; se requiere un programa que dadas las dimensiones (largo y ancho)

de la ventana y de la pared muestre en la pantalla el tiempo necesario para pintar la pared si se sabe que se puede pintar 1 mt2

en 10 min. ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Sobrecarga

En C++ es posible tener varios métodos dentro de la misma clase con el mismo nombre, a esto se le conoce como sobrecarga.

Es posible sobrecargar un método declarando otro con el mismo nombre pero diferente número o tipo de parámetros. C++ sabe

cual método debe ejecutarse utilizando la cantidad y tipo de parámetros que se colocan al mandar llamar a dicho método.

La sobrecarga no es exclusiva de funciones miembro de una clase, también es posible sobrecargar funciones libres; es decir,

funciones que no pertenecen a ninguna clase.

Constructores

Es muy común que algunas partes del objeto requieran ser inicializadas antes de poder ser utilizadas. C++ permite que los objetos

sean inicializados cuando se crean, a través de un constructor. Un constructor es una función especial que es miembro de la

clase, que tiene el mismo nombre de la clase y que se manda llamar automáticamente cuando se crea una instancia u objeto de

una clase.

En C++ un constructor tiene las siguientes características:

- Su nombre es el mismo que el de la clase
- No tiene definido un valor de retorno

Un constructor puede o no tener parámetros, al constructor sin parámetros se le conoce como el constructor Default.

Cuando se declara una instancia de una clase que tiene un constructor con parámetros, para que se llame a dicho constructor se

requiere el siguiente **formato:**

NombreClase NombreVariable(listaParametros);

donde listaParametros son los parámetros actuales que se enviarán a los parámetros formales del constructor que se quiere llamar.

Por ejemplo, supongamos que vamos a declarar dos constructores para la clase Circulo como se muestra a

continuación:

```
Circulo::Circulo()
{
    radio = 0;
}

Circulo::Circulo(int ra)
{
     radio = ra;
}
```

podemos declarar instancias de la clase Círculo de la siguiente forma:

```
Circulo a;         // se llama al constructor default, porque no se especifícan
parámetros.

Circulo b(10);    // se llama al constructor que recibe como parámetro el
radio, porque es el que   coincide con los parámetros que se colocan al crear
el objeto.
```

Nota que si se va a "llamar" el constructor default, al declarar la variable no se colocan los paréntesis.

Si el programador no crea un constructor explicitamente, C++ crea uno "vacío" que se encarga de realizar algunas acciones

necesarias al crear un objeto.

Destructores

Un destructor es una función que se ejecutada automáticamente cuando se destruye un objeto, por ejemplo cuando un objeto es

local a un método y se termina la ejecución de dicho método. El destructor por si mismo no destruye el objeto, pero es útil para

liberar memoria dinámica que se había reservado al crear el objeto, o para cerrar un archivo que se abrió al crear el objeto, etc.

Un destructor en C++ tiene las siguientes características:

- Tiene el mismo nombre de la clase, pero precedido por el caracter ~.
- No tiene definido un valor de retorno

Una clase puede tener solamente un destructor, el cual no debe recibir parámetros ni tener valor de retorno. Si el programador no

crea un destructor C++ crea uno "vacío" que se encarga de realizar algunas acciones necesarias al destruir un objeto.

Ejemplo completo

El siguiente ejemplo muestra la declaración de una clase (la clase Pila) y una aplicación que hace uso de ella.

```
#include <iostream.h>

const int SIZE = 100;

//Definición de la clase Pila
class Pila {
         int pila[SIZE];
         int tope;
    public:
         Pila(); // constructor
         ~Pila(); // destructor
         void push(int i);
         int pop();
       };

// constructor de la clase pila
Pila::Pila()
{
    tope = 0;
    cout<<"Se inicializa la pila"<<endl;   // este letrro se utiliza solo para
entender el ejemplo
}


// destructor de la clase pila
Pila::~Pila()
{
    cout<<"se destruye la pila"<<endl;   // este letrero se utiliza solo para
hacer más claro el ejemplo
}


void Pila::push(int valor)
{
    if (tope ==SIZE)
    {
        cout<<"La pila esta llena"<<endl;
        return;
    }
    pila[tope] = valor;
    tope++;
}

int Pila::pop()
{
    if (tope == 0)
    {
        cout<<"No hay nada en la pila"<<endl;
        return -1;
    }
    tope--;
    return pila[tope];
}


int main()
{
    Pila a, b;

    a.push(1);
```

```
    b.push(2);
    a.push(3);
    b.push(4);

    cout<<a.pop()<<endl;
    cout<<a.pop()<<endl;
    cout<<b.pop()<<endl;
    cout<<b.pop()<<endl;

    return 0;
}
```

Ejercicio

Copia los ejemplos que se incluyen en el material y pruébalos.

Define la clase Fecha, de acuerdo con las siguientes especificaciones:

Atributos:

- dia
- mes
- año

Constructores:

- que inicialice con una fecha fija que tu definas
- que reciba como parámetro los valores para inicializar la fecha

Métodos públicos:

- que permita modificar el valor de la fecha
- que muestre en la pantalla la fecha usando el formato dia / mes / año
- que muestre en la pantalla la fecha poniendo el mes con palabras
- que permita verificar si una fecha es válida; este método debe ser utilizado por el constructor y el método que modifica el

  valor de la fecha, si el usuario trata de inicializar con una fecha inválida se inicializará con el valor fijo que tu hayas definido.

Realiza después una aplicación para probar tu clase, debe al menos crear 2 objetos de tipo Fecha, utilizando cada uno de los

constructores y después mostrar las fechas correspondientes en la pantalla. ver solución

Ligas sugeridas
http://www.cplusplus.com/doc/tutorial/ http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Módulo 5: Arreglos

1. Arreglos de una Dimensión

¿Que es un arreglo?

Un arreglo es un tipo de dato estructurado que permite guardar colecciones de elementos del mismo tipo.

Declaración de arreglos

Para declarar un arreglo se utiliza el siguiente formato:

tipo nombre_arreglo [tamaño];

donde tipo es el tipo de los datos que almacenará el arreglo. Es importante mencionar que se pueden declarar arreglos de los tipos

primitivos de C++, o bien de tipos definidos por el usuario.

Tamaño representa la cantidad de casillas que se reservan para el arreglo. En C++ todos los arreglos empiezan en el subíndice 0

y llegan al subíndice tamaño-1.

Por ejemplo:

```
int arr[6];        // arreglo de 6 elementos enteros, los subíndices van del 0 al 5

char cad[10];   // arreglo de 10 elementos de tipo caracter, los subíndices van del 0 al 9.
```

Otra forma de declarar arreglos:

También es posible declarar un arreglo declarando primero un tipo con typedef de la siguiente forma:

```
typedef tipo nombre_tipo[tamaño];   // declaración del tipo

nombre_tipo Nombre_variable;        // declaración de la variable de tipo arreglo
```

Por ejemplo:

```
typedef int arr8[8];
arr8 lista;
```

En este caso se declara el tipo de dato arr8 que es un arreglo de 8 elementos enteros, cuyos subíndices van del valor 0 al 7. En el

segundo renglón de la declaración se declara el arreglo del tipo arr8.

Uso de los elementos del arreglo

Para usar los elementos individuales de un arreglo se usa el siguiente formato:

arreglo[subíndice]

Como un elemento de un arreglo es un dato, se puede usar como cualquier variable de ese tipo; Enseguida se pueden ver algunos ejemplos:

```
int arr[4];
arr[3] = 8;
cin>>arr[2];
cout<<arr[3];
arr[0] = arr[1] + arr[2];
int k = 2;
arr[k+1] = 20;
```

Ejemplo:

```
#include <iostream.h>

int main()
{
    int k, arreglo[20];

    // coloca en cada casilla el valor de su subíndice
    for (k = 0; k < 20; k++)
         arreglo[k] = k;

    // muestra el contenido del arreglo
    for (k = 0; k < 20; k++)
         cout<<"Casilla "<<k<<" "<<arreglo[k]<<endl;
    return 0;
}
```

Inicializar arreglos en la declaración

En C++ es posible inicializar un arreglo al declararlo; esto se hace colocando un operador de asignación y después entre llaves la

lista de valores para el arreglo separados por comas, veamos los siguientes ejemplos:

double arreglo[3] = { 23.5, 54.22, 78};

char cadena[4] = {?a?, ?b?, ?c?, ?d?};

Ahora veamos el siguiente ejemplo:

int valores[20] = {2, 4, 6};

En este ejemplo se muestra el caso de no definir valores para todos los elementos del arreglo, en este caso los elementos

faltantes se inicializan con 0 (cero) o con el caracter nulo (\0) dependiendo del tipo de dato del arreglo.

También es posible tener este caso:

int arreglo[ ] = { 2, 4, 6, 8};

En este caso, la cantidad de datos determina el tamaño del arreglo que se está declarando.

Ejemplo completo:

```
// Ejemplo de arreglos de una dimensión
// Método de ordenamiento de burbuja
#include <iostream.h>

int main()
{
   const int tamArr = 10;
   int lista[tamArr] = {22, 7, 14, 9, 2, 80, 40, 67, 51, 38};
   int tmp;

   // muestra el arreglo original
   for (int i = 0; i < tamArr; i++)
      cout<<lista[i]<<"\t"; //muestra el número y un tab

   //método de ordenamiento de burbuja
   // para controlar la cantidad de pasadas
   for(int pasada = 0; pasada < tamArr-1; pasada++)
      // para controlar el número de comparaciones por pasada
      for (int k = 0; k < tamArr-1; k++)
         if (lista[k] > lista[k+1])
         {
            tmp = lista[k];
            lista[k] = lista[k+1];
            lista[k+1] = tmp;
         }

   // muestra el arreglo ordenado
   cout<<"Los datos ordenados son"<<endl;
   for (int j = 0; j < tamArr; j++)
      cout<<lista[j]<<"\t";

   return 0;
}
```

Ejercicio

Copia los ejemplos que se incluyen en el material y pruébalos.

Realiza un programa que pida al usuario una lista de números de tipo double y encuentre el menor y su posición y las muestre en la pantalla. ver solución

Ligas sugeridas
http://www.cplusplus.com/doc/tutorial/ http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Declaración de arreglos multidimensionales

En C++ es posible tener arreglos multidimensionales. Para definir un arreglo multidimensional utiliza el siguiente formato:

tipo Nom_var[tam1][tam2]?[tamN];

y para accesar un elemento de un arreglo multidimensional se utiliza:

Nom_var[sub1][sub2]?[subN];

Nota que cada subíndice se encierra en un par de corchetes([ ]).

El uso más común de los arreglos multidimensionales es para representar arreglos de 2 dimensiones o tablas, en donde los datos

están acomodados por renglones y columnas. Por convensión el primer subíndice se refiere a los renglones y el segundo a las

columnas.

Por ejemplo podemos definir un arreglo de elementos enteros de 2 dimensiones con 3 renglones y 4 columnas usando la siguiente

instrucción:

int lista[3][4];

y por ejemplo para accesar el elemento que se encuentra en el renglón 0 y la columna 2 de este arreglo utilizamos la variable:

lista[0][2];

Aunque no es muy común, también podemos definir arreglos de más de 2 dimensiones, por ejemplo, podemos definir un arreglo de

elementos enteros de 3 dimensiones con 4 renglones, 3 columnas y 2 niveles de profundidad usando la siguiente línea.

int cubo[4][3][2];

También es posible declarar el arreglo declarando primero un tipo con typedef de la siguiente forma:

typedef tipo nombre_tipo[tamaño1][tamaño2]; // declaración del tipo

nombre_tipo Nombre_variable; // declaración de la variable de tipo arreglo

Por ejemplo:

typedef double tabla[8][4];

tabla lista;

En este caso se declara el tipo de dato tabla que es un arreglo de 2 dimensiones de elementos de tipo double, cuyos subíndices

van del valor 0 al 7 y del 0 al 3. En el segundo renglón de la declaración se declara el arreglo del tipo tabla.

En C++ un arreglo de 2 dimensiones es en realidad un arreglo de arreglos. Por ejemplo si declaramos el arreglo

int arr[10][5];

en realidad estamos declarando un arreglo de 10 elementos, cuyos elementos son arreglos de 5 elementos de tipo entero. Este es

la manera en que internamente se maneja el arreglo y el programador no tendría porque enterarse de ello; aquí lo mencionamos

porque esto nos ayudará a entender algunas características de los arreglos de 2 dimensiones.

Inicializar arreglos en la declaración

Cuando se requiere inicializar en la declaración un arreglo de 2 dimensiones es útil recordar que en C++ internamente un arreglo de

dos dimensiones es un arreglo de arreglos, entonces se debe dar el valor para cada uno de los arreglos (como una lista de

elementos encerrados entre llaves y separados por comas) como se muestra en el siguiente ejemplo:

int lista[3][2] = {{1,2},{3,4},{5,6}};

Ejemplo:

```
// Programa que encuentra el valor mayor de la matriz, y ademas dice
// en que renglon y en que columna se encuentra dicho valor.
#include <iostream.h>

typedef int Matriz[20][20];
```

```
int main()
{
   Matriz A;           // Variable de tipo arreglo de 2 dimensiones
   int N, M;           // Dimensiones de la matriz
   int ren, col;       // variables auxiliares para las búsquedas
   int mayor;         // variable en la que se guardará el valor mayor
   int renM, colM;   // variables que guardarán el reng y la col en las
                        // que se encontró el valor mayor

   cout<<"Cuantos renglones tiene la matriz? ";
   cin>>N;
   cout<<"Cuantas columnas tiene la matriz? ";
   cin>>M;

   // pide los datos para la matriz
   for (ren=0; ren < N; ren++)
      for (col=0; col < M; col++)
      {
            cout<<"Teclea el valor para el elemento "<<ren<<","<<col<<" ";
            cin>>A[ren][col];
      }

   // muestra la matriz
   for (ren=0; ren<N; ren++)
   {
      for (col=0; col<M; col++)
         cout<<A[ren][col]<<"\t";
      cout<<endl;
   }

   //Busca el valor mayor de la matriz

   mayor = A[0][0];
   renM = 0;
   colM = 0;
   for (ren = 0; ren < N; ren++)
      for (col = 0; col < M; col++)
         if (A[ren][col] > mayor)
         {
            mayor = A[ren][col];
            renM = ren;
            colM = col;
         }
   cout<<"El valor mayor es "<<A[renM][colM]<<endl;
   cout<<"y se encuentra en las posiciones "<<endl;
   cout<<"renglon "<<renM<<" y columna "<<colM<<endl;

   return 0;
}
```

Ejercicio

Realiza un programa que pida al usuario las dimensiones deseadas para un arreglo de dos dimensiones y luego le pida los datos para

llenar el arreglo. Después el programa debe dar la opción de calcular la suma de cualquier renglón o columna del arreglo. Tu

programa debe trabajar con arreglos que tengan como cantidad máxima de renglones y/o columnas 10. ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Uso de arreglos de una dimensión como parámetro

Para declarar un arreglo como parámetro formal de una función se utiliza:

tipo nom_parametro[ ];

No es necesario especificar el tamaño del arreglo, sin embargo, al programar la función es necesario saberlo; por lo que es muy

común agregar un parámetro de tipo entero en el que se envía la cantidad de casillas ocupadas del arreglo.

Por default, un arreglo se considera un parámetro por referencia (de entrada y/o salida); es decir, cuando se recibe un arreglo

como parámetro y dentro de la función se modifica alguno de sus valores, el parámetro actual que se usó al mandar llamar a dicha

función se modifica; por otro lado, si se desea que el arreglo actúe como un parámetro por valor (de entrada); es decir, que no se

modifique dentro de la función se debe anteponer la palabra const a la declaración del parámetro; esto es en realidad una

precaución con la que se está avisando al compilador que esa función no debe modificar el valor de la variable; en caso de que por

error en la función se tratara de modificar el valor de alguna casilla del arreglo el compilador debe enviar un error indicándolo.

Un arreglo no puede ser regresado como valor de retorno de una función; por lo que si se requiere una función que regrese un

arreglo, éste se debe especificar como parámetro.

A continuación se encuentra un ejemplo en el que se incluye una función que recibe como parámetro un arreglo de precios, su

tamaño y un incremento, la función regresa el arreglo con los precios nuevos, es decir, después de aplicarles el incremento

indicado en el tercer parámetro:

```
#include <iostream.h>

void incrementa(double precios[], int tam, double inc)
{
    for (int i = 0; i < tam; i++)
        precios[i] = precios[i] * (1 + inc);
}

void lee(double arr[], int tam)
{
    for (int i = 0; i < tam; i++)
    {
        cout<<"Teclea el valor "<<i<<" ";
        cin>>arr[i];
    }
}


void muestra(const double arr[], int tam)
{
    for (int i = 0; i < tam; i++)
        cout<<"precio "<<i<<" "<<arr[i]<<endl;
}

int main()
{
    double precios[20];
    int tam;
    double incremento;

    cout<<"Cuantos precios hay en la lista?"<<endl;
    cin>>tam;
    cout<<"Teclea los precios "<<endl;
    lee(precios, tam);

    cout<<"cual es el incremento para los precios "<<endl;
    cout<<"teclea un valor entre 0 y 1 ";
    cin>>incremento;
    incrementa(precios, tam, incremento);

    // muestra la tabla de precios con el incremento
    muestra(precios, tam);

    return 0;
}
```

Uso de arreglos multidimensionales como parámetro

Cuando se requiere un arreglo de 2 o más dimensiones como parámetro no se requiere especificar el tamaño de la primera

dimensión; pero si se requiere el tamaño de las siguientes dimensiones. Como la primera dimensión no es necesaria, normalmente se

agrega un parámetro adicional de tipo entero que contiene la cantidad de datos de la primera dimensión del arreglo.

Revisemos el siguiente ejemplo en el que aparece como parámetro un arreglo de una dimensión y de dos dimensiones en la función

CopiaCol:

```
#include <iostream.h>

//Funcion que copia la columna A de la matriz al arreglo Arr
void CopiaCol(int Mat[][10], int orden, int A, int Arr[])
{
    for(int i=0; i<orden; i++)
        Arr[i] = Mat[i][A];
}

//Funcion que pide el orden de la matriz y luego lee sus elementos
void LeeMat(int Mat[][10], int &orden)
{
    cout<<"Cual es el orden de la matriz? ";
    cin>>orden;
    for (int ren=0; ren < orden; ren++)
        for (int col=0; col < orden; col++)
        {
            cout<<"Teclea el valor para el elemento "<<ren<<","<<col<<" ";
            cin>>Mat[ren][col];
        }
}

//Funcion que muestra los elementos de la matriz en la forma tradicional
void MuestraMat(int Mat[][10], int orden)
{
    for (int ren=0; ren<orden; ren++)
    {
        for (int col=0; col<orden; col++)
            cout<<Mat[ren][col]<<"\t";
        cout<<endl;
    }
}

//Funcion que muestra los elementos del arreglo
void MuestraArr(int Arr[], int orden)
{
    for (int ren=0; ren<orden; ren++)
        cout<<Arr[ren]<<"\t";
}

int main()
{
    int Matriz[10][10];
    int A, orden;
    int Arreglo[10];

    LeeMat(Matriz, orden);
    MuestraMat(Matriz, orden);
    cout<<"Copiara la columna A, Cual es el valor de A?";
    cin>>A;

    CopiaCol(Matriz, orden, A, Arreglo);
    cout<<"muestra el arreglo que tiene la columna "<<A<<" de la
matriz"<<endl;
    for (int i=0; i<orden; i++)
        cout<<Arreglo[i]<<"\t";
    cout<<endl;

    return 0;
```

```
}
```

Ejercicio

Realiza un programa que pida al usuario el orden (es decir, las dimensiones) de una matríz cuadrada, después que pida los valores

para las casillas del arreglo y después que muestre en la pantalla los valores que se encuentran en las 2 diagonales de la matriz.

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Apuntador

La computadora guarda las variables en la memoria y cada espacio de memoria tiene una dirección. Normalmente no necesitamos saber la dirección en la que está guardada una variable, ya que nos referimos a ella por su nombre.

En C++ existen variables que puede guardar una dirección de memoria, se llaman apuntadores y una variable de tipo apuntador puede guardar direcciones de variables de un tipo determinado; es decir, hay apuntadores a entero, apuntadores a double, etc.

Para declarar un apuntador se utiliza el siguiente formato:

```
tipo *nombreVariable;
```

Por ejemplo:

```
int *ptr1;      // apuntador a entero
double *ptr2; // apuntador a double
char *ptr3;    // apuntador a caracter
```

Nota que el * puede ir pegado al tipo de dato o a la variable.

Si escribes la siguiente expresión en un programa de C++:

```
ptr1 = ptr2;
```

el compilador marca un error porque ptr1 y ptr2 son apuntadores de diferente tipo.

En la siguiente declaración:

```
int *p, n;
```

p es una variable de tipo apuntador a entero, mientras que n es una variable de tipo entero.

Operador dirección &

El operador & se utiliza para obtener la dirección de una variable y se puede usar con el siguiente formato:

```
&nombreVar;
```

Por ejemplo:

```
int *p, n;     // declaro p como un apuntador a entero y n como un entero
p = &n;        // asigno la dirección de la variable n en la variable p
```

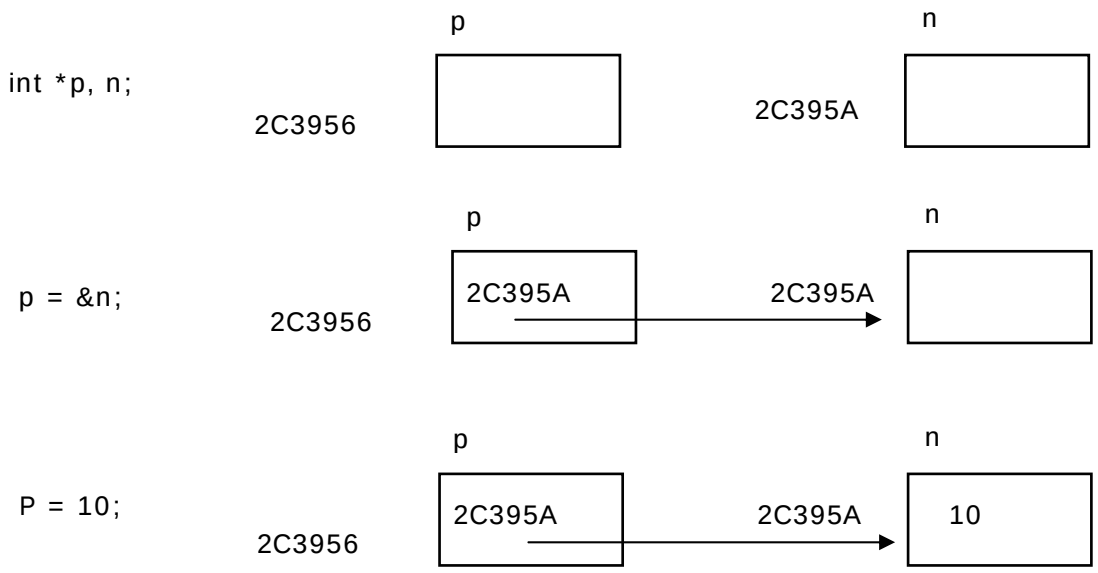
Operador *

El operador * aplicado a una variable de tipo apuntador genera la variable a la cual apunta, por ejemplo:

```
int *p, n;
p = &n;
```

En este ejemplo la expresión ***p** se refiere a la variable n.

Veamos ahora este ejemplo gráfico:



El valor NULL (nulo)

Para indicar que una variable no apunta a nada se utiliza el valor Nulo (NULL) como se muestra en el siguiente ejemplo:

```
int *p;
p = NULL;
```

Desplegar un apuntador

Es posible mostrar el contenido de una variable de tipo apuntador en la pantalla utilizando un cout como se muestra en el siguiente ejemplo:

```
int main()
{
  int x, y, *p, *q;
```

```
  p = &x;
  *p = 5;
  q = &y;
  *q = 23;
  cout<<*p<<" "<<*q<<endl;
  q = p;
  *p = 35;
  cout<<*p<<" "<<*q<<endl;
  q = NULL;
  cout<<x<<" "<<y<<endl;
  return 0;
}
```

Si ejecutamos el siguiente programa mostrará en la pantalla lo siguiente:

```
5 23
35 35
35 23
```

Ejercicio

Copia el ejemplo que se incluye en el material y pruébalo.

Sigue el programa que se muestra a continuación e indica qué valores se mostrarán en la pantalla

```
#include <iostream.h>
int main()
{
   int x, y, *p, *q;
   p = &x;
   *p = 12;
   q = &y;
   *q = 23;
   cout << *p <<" "<<*q<<endl;
   *p = *q;
   cout << *p <<" "<<*q<<endl;
   q = NULL;
   cout<<*p<<endl;
   cout<<x<<" "<<y<<endl;
   return 0;
}
```

ver solucion

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Relación entre Apuntadores y Arreglos

Si se declara el siguiente arreglo

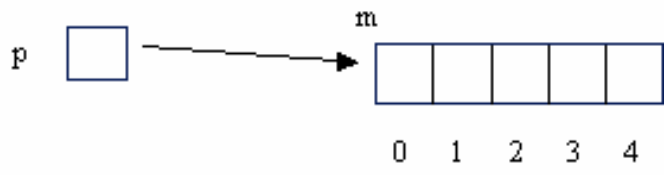
```
int m[5];
```

internamente C++ declara m como un apuntador a entero y coloca ahí la dirección del primer elemento del arreglo.

Supongamos que tenemos lo siguiente:

```
int *p;
p = m;
```

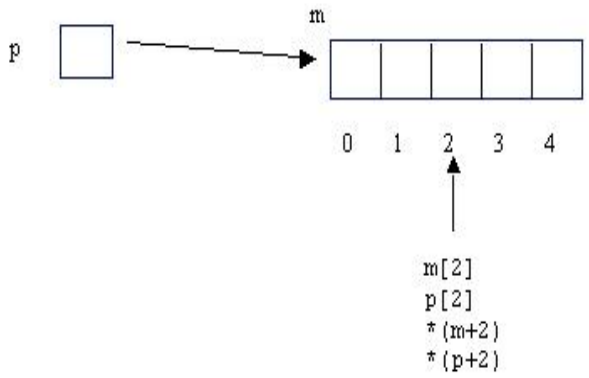
gráficamente lo podemos representar así:



y para accesar los elementos del arreglo también se puede usar el apuntador de cualquiera de las siguientes maneras:

m[2]	se refiere al elemento de la casilla 2 del arreglo
p[2]	se refiere al elemento de la casilla 2 del arreglo
*(p+2)	se refiere al elemento que está 2 posiciones enteras más allá del inicio del arreglo, lo cual es equivalente a decir que es el elemento de la casilla 2 del arreglo
*(m+2)	se refiere al elemento que está 2 posiciones enteras más allá del inicio del arreglo, lo cual es equivalente a decir que es el elemento de la casilla 2 del arreglo

es decir:



En la expresión *(p+2) al sumar 2 al apuntador p, no se está sumando el número 2 sino que se está avanzando 2 posiciones en la memoria; es decir, p es un apuntador a entero, por lo que al sumarle 2 se avanza el espacio de memoria necesario para avanzar 2 posiciones enteras.

El paréntesis es necesario porque el operador * tiene una prioridad más alta que el operador +.

Aritmética de apuntadores

Cada vez que un apuntador se incrementa, apunta a la localidad de memoria del siguiente elemento de su tipo base (o sea, del tipo del que fue definido).

Cada vez que se decrementa, apunta a la localidad de memoria del elemento anterior. Esta forma de trabajar asegura que siempre un apuntador apunta a un elemento apropiado de su tipo base.

Por ejemplo, supón que tienes la siguiente declaración y que un entero ocupa 2 bytes:

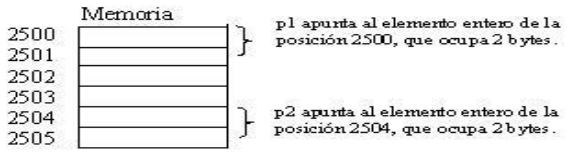
```
int *p1, *p2;
```

y supón que p1 contiene la dirección de memoria 2500, al ejecutar la operación:

```
p2 = p1 + 2;
```

la memoria queda de la siguiente manera

Memoria



Verificación de los limites del arreglo

En base a lo que se ha explicado en esta sección, se puede entender porque C++ no hace verificación de los límites de un arreglo; es decir, si se tiene el siguiente caso:

```
int m[5];
m[15] = 10;
```

se está asignando el valor 10 a una posición de memoria que no pertenece al arreglo (porque el arreglo m tiene casillas de la 0 a la 4); entonces, es responsabilidad del programador verificar que esto no ocurra.

Ejercicio

Supón que tienes declarado el siguiente arreglo:

```
int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
```

dí que muestra en la pantalla el siguiente segmento de código:

```
int *p;

p = arr;
*p = 20;
*(p + 2) = 40;
*(arr + 5) = 60;

for (int i =0; i<10; i++)
   cout<<arr[i]<<"\t";
```

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Strings terminados en nulo.

C++ soporta 2 tipos de strings. El primero de ellos son los strings terminados en nulo; el otro tipo es utilizando la clase String, que te da una forma orientada a objetos para maejar los strings. En esta sección nos enfocaremos a los strings terminados en nulo.

Nota. Dependiendo de la implementación del compilador, la biblioteca estándar de C++ podría llamarse string.h, o bien, cstring.h.

Declarar un string

Un string se representa internamente como un arreglo de caracteres; por lo tanto se declara de esta forma con el siguiente formato:

char nomVar[tamaño+1]

C++ agrega un caracter nulo (\0) al final de un string para indicar en dónde termina. Esa es la razón por la que se debe declarar reservando un caracter más de lo que se requiere guardar.

Por ejemplo, podemos declarar el string Palabra que contenga espacio para guardar un string de hasta 15 caracteres de la siguiente forma:

char Palabra[16];

y se puede utilizar un valor string para inicializar el string en su declaración de la siguiente forma:

char Palabra[15] = "Clase";

este string internamente se representa así:

Palabra

C	l	a	s	e	\0									
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Es decir, se coloca un caracter en cada una de las casillas del arreglo, y después del último caracter se agrega el caracter nulo (nota que el segundo renglón de esta tabla contiene el número correspondiente a la casilla del arreglo).

Cuando usas un valor string en el programa, es decir una lista de caracteres encerrados entre comillas, en memoria se crea un string terminado en nulo; es decir, siempre una constante string se maneja como un string terminado en nulo.

Como el string es en realidad un arreglo de caracteres, podemos accesar cada uno de los caracteres de la siguiente forma:

Nombre_string[posición]

Por ejemplo, podemos accesar el primer caracter del string Palabra utilizando:

Palabra[0]

Mostrar en la pantalla un string

Se puede mostrar un string en la pantalla utilizando cout de la forma tradicional.

Por ejemplo:

cout<<Palabra;

Leer un string

Se puede leer un string del teclado utilizando el cin de la forma tradicional. Solo es importante recordar que el cin ignora los espacios en blanco, tab o return del inicio y termina de leer cuando se encuentra con espacio en blanco, tab o return; por lo que con un cin solamente se puede leer valores String que contengan una sola palabra.

Por ejemplo:

cin>>Palabra;

También se puede leer un string del teclado utilizando getline, la diferencia con cin es que getline no ignora los espacios en blanco ni los tab y termina de leer cuando se encuentra con un Return; por lo que con getline sí es posible leer un string que contenga varias palabras.

El formato para utilizarlo es:

cin.getline(nombreString, longitud);

esta función toma del teclado cuando mucho longitud ?1 caracteres y termina de leer cuando encuentra un Return; el string leido se coloca en la variable nombreString. Este estatuto retira el caracter \n del input stream, pero no lo pone en el string.

Por ejemplo:

cin.getline(Palabra, 15);

Algunas funciones existentes en la biblioteca string.h

C++ contiene una gran variedad de funciones de manejo de strings, en esta sección se explicarán las más comunes (en algunas versiones de C++ la biblioteca podría ser cstring.h).

Función para obtener la longitud del string

Formato:

variableEntera = strlen(NombreString);

Esta función regresa la longitud de un string. El valor que regresa es el número de caracteres que

contiene el string sin contar el caracter nulo de terminación.

Por Ejemplo:

```
char Pal[10] = ?hola?;
int x = strlen(Pal);
```

En este ejemplo la variable x toma el valor 4.

Función para concatenar dos strings

Formato:

```
strcat(primerString, segundoString);
```

Esta función recibe como parámetro dos strings, y concatena el contenido del segundo string al final del primero. La longitud del string resultante es igual a la suma de las longitudes de los strings concatenados.

Por ejemplo:

```
char s1[10] = ?uno?;
char s2[10] = ?dos?;
strcat(s1, s2);
```

Al ejecutar estos estatutos la variable s1 toma el valor ?unodos?

Función para copiar un string a otro

Formato:

```
strcpy(primerString, segundoString);
```

Esta función copia el contenido de un string en otro. A diferencia de la concatenación, esta función borra el contenido del primer string y en su lugar pone el contenido del segundo.

Por ejemplo:

```
char s1[10] = ?uno?;
char s2[10] = ?dos?;
strcpy(s1, s2);
```

Al ejecutar estos estatutos la variable s1 toma el valor "dos"

No es posible asignar directamente el valor de una constante string a una variable de tipo string. Para hacerlo es necesario utilizar la función strcpy poniendo como segundo sting el valor a copiar.

Por ejemplo:

```
char Pal[10];
strcpy(Pal, ?algo?);
```

Después de ejecutar estos estatutos la variable Pal toma el valor ?algo?

Función para comparar dos strings

Formato:

varEntera = strcmp(primerString, segundoString);

Esta función realiza una comparación entre dos strings empezando por el primer caracter de cada string y continuando la comparación caracter por caracter (mientras sean iguales) hasta encontrar que dos caracteres difieran, o bien, que se alcance el final de alguno de los strings. El resultado de la comparación lo regresa de acuerdo con la siguiente tabla:

Si el primer string es	strcmp regresará un valor
menor que el segundo	< 0
igual que el segundo	== 0
mayor que el segundo	> 0

Por ejemplo:

```
char s1[10] = ?uno?;
char s2[10] = ?dos?;
int x = strcmp(s1, s2);
```

Al ejecutar estos estatutos la variable entera x tomará un valor mayor a 0

Ejemplo:

```
#include <iostream.h>
#include <string.h>

// Esta funcion verifica si un string es un palíndrome
// regresa 1 si es palíndrome y 0 si no lo es
int esPalindrome(char str[])
{
   int i,j;
   char aux[20];

   // construye un string que contiene los caracteres al revés
   for (i = strlen(str)-1, j=0; i>=0; i--, j++)
       aux[j]=str[i];
   aux[j]='\0'; // al construir el string no olvidar el nulo al final

   if (strcmp(aux, str) == 0)
      return 1;
   else
      return 0;
}

int main()
{
    char palabra[20], resp;

    do
```

```
    {
        cout<<"Teclea una palabra"<<endl;
        cin>>palabra;

        if (esPalindrome(palabra))
            cout<<"si es palindrome"<<endl;
        else
            cout<<"no es palindrome"<<endl;
        cout<<"Revisar otra palabra"<<endl;
        cin>>resp;
    } while ((resp == 's') || (resp == 'S'));
    return 0;
}
```

Ejercicio

Realiza un programa que pida al usuario una palabra y 2 letras. El programa debe sustituir todas las ocurrencias de la primera letra por la segunda letra; por ejemplo:

Si se teclea la palabra abecedario y la primera letra es a y la segunda z se obtendrá el siguiente string:

zbededzrio
ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Arreglos con objetos

Hay dos formas en las que se pueden relacionar los arreglos con los objetos:

- Un arreglo puede contener elementos de tipo objetos.
- Una clase puede contener como atributos variables de tipo arreglo.

En esta sección veremos el caso en el que tenemos un arreglo con elementos de tipo Objeto.

Supón que tenemos la siguiente declaración:

```
class Articulo
{
   public:
      Articulo();
      void inicializa(int, int, double);
      double getPrecio();
      int getClave();
      int getExistencia();
      void registraEntrada(int cant);
      int registraSalida(int cant);
   private:
      int clave;
      int existencia;
      double precio;
};

Articulo Lista[20];
```

Para asignar valor al primer artículo del arreglo, usamos:

```
   Lista[0].inicializa(12398, 200, 47.50);
```

Nota que aquí se combina la sintaxis del uso de arreglos y del uso de objetos. Es importante que notes que lo que viene después del punto debe ser algún miembro público del objeto; es decir, no puedes accesar directamtente los datos del objeto que son privados.

Ejemplo de programa

```
// Esta clase se encuentra en el archivo Articulo.h
// Clase articulo.
// Cada articulo esta definido por su clave, precio y la cantidad de articulos en existencia.
#include <iostream.h>

class Articulo
{
   public:
      Articulo();
```

```
        void inicializa(int, int, double);
        double getPrecio();
        int getClave();
        int getExistencia();
        void registraEntrada(int cant);
        int registraSalida(int cant);
    private:
        int clave;
        int existencia;
        double precio;
};

Articulo::Articulo()
{
    clave = 0;
    existencia = 0;
    precio = 0;
}

// pone al objeto los valores recibidos como parámetro
void Articulo::inicializa(int cve, int exis, double pre)
{
    clave = cve;
    existencia = exis;
    precio = pre;
}

// Regresa el precio
double Articulo::getPrecio()
{
    return precio;
}

// Regresa la clave
int Articulo::getClave()
{
    return clave;
}

// Regresa la cantidad en existencia
int Articulo::getExistencia()
{
    return existencia;
}

// Agrega cant a la cantidad de articulos en existencia
void Articulo::registraEntrada(int cant)
{
    existencia += cant;
}

// Si no hay suficientes articulos regresa 0, si hay suficientes
// registra la salida y regresa 1
int Articulo::registraSalida(int cant)
{
    if (cant > existencia)
```

```
      return 0; // false indica que no hubo suficientes artículos
   else
   {
      existencia -= cant;
      return 1; // true indica que si registro la salida
   }
}


// Este programa se encuentra en el archivo UsaArt.cpp
#include <iostream.h>
#include "Articulo.h"

Articulo lista[30];
int tam;

void verInventario()
{
   int i;

   cout<<"La lista de artículos en el almacén es la siguiente"<<endl;
   cout<<"\t"<<"Clave"<<"\t"<<"Cantidad en Existencia"<<"\t"<<"Precio"<<endl;
   for (i = 0; i < tam; i++)
      cout<<"\t"<<lista[i].getClave()<<" "
            <<"\t \t \t"<<lista[i].getExistencia()<<" "
            <<"\t $"<<lista[i].getPrecio()<<endl;
   cout<<endl<<endl;
}

int Busca(int cve)
{
   int i;
   int posicion = -1; // -1 indica que no se encontró el artículo
   for (i = 0; i<tam && posicion == -1; i++)
      if (lista[i].getClave() == cve)
         posicion = i; // posición en la que encuentra el artículo
   return posicion;
}

void registraLlegada()
{
   int posicion, cve, cantidad;

   cout<<endl;
   cout<<"cuál es la clave del artículo"<<endl;
   cin>>cve;
   //busca la clave en el almacen
   if ((posicion = Busca(cve)) != -1)
   {
      cout<<"cuantos artículos llegaron"<<endl;
      cin>>cantidad;
      lista[posicion].registraEntrada(cantidad);
   }
   else
      cout<<"Esa clave no existe "<<endl;
   cout<<endl;
```

```
}

void registraSalida()
{
   int posicion, cve, cantidad;

   cout<<endl;
   cout<<"cuál es la clave del artículo"<<endl;
   cin>>cve;
   //busca la clave en el almacen
   if ((posicion = Busca(cve)) != -1)
   {
      cout<<"cuantos artículos salieron"<<endl;
      cin>>cantidad;
      if (!lista[posicion].registraSalida(cantidad))
         cout<<"No hay suficientes artículos "<<endl;
   }
   else
      cout<<"Esa clave no existe "<<endl;
   cout<<endl;
}

char menu()
{
   char opc;
   cout<<endl<<endl;
   cout<<" Menu de Opciones"<<endl;
   cout<<"a. Ver el inventario"<<endl;
   cout<<"b. Registrar una llegada al almacén"<<endl;
   cout<<"c. Registrar una salida de almacén"<<endl;
   cout<<"d. Salir del programa"<<endl;
   do
   {
      cout<<"Opcion ->";
      cin>>opc;
   } while ((opc < 'a') || (opc > 'd'));
   return opc;
}

int main()
{
   int i, cve, exis;
   double pre;
   char opc;

   cout<<"Cuantos artículos ";
   cin>>tam;
   for (i = 0; i < tam; i++)
   {
      cout<<"Teclea la clave para el artículo "<<i+1<<endl;
      cin>>cve;
      cout<<"Teclea la existencia para el artículo "<<i+1<<endl;
      cin>>exis;
      cout<<"Teclea el precio para el artículo "<<i+1<<endl;
      cin>>pre;
      lista[i].inicializa(cve, exis, pre);
```

```
   }
   do
   {
      opc = menu();
      switch (opc)
      {
         case 'a': verInventario();
                   break;
         case 'b': registraLlegada();
                   break;
         case 'c': registraSalida();
                   break;
      }
   } while (opc != 'd');
   return 0;
}
```

Ejercicio

Copia el ejemplo que se incluye en el material y pruébalo.

Diseña la clase Alumno en base a la siguiente especificación:

Atributos:

- id
- calificacion

Métodos:

- setId
- setCalificación
- getId
- getCalificación

Realiza después un programa que cree un arreglo de objetos de tipo alumno, que pida al usuario una lista de alumnos y después muestre en la pantalla la lista de todos los alumnos y el promedio de calificaciones de los alumnos. ver solucón

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Empecemos revisando un ejemplo: La clase Fracción

```
class Fracción {
   public:
      Fraccion(int, int);       //Inicializa con los valores que recibe
      Fraccion();               //Inicializa con el valor 1/1
      void Muestra();           //Muestra con el formato num/den
   private:
      int Num, Den;
};

// Constructor que inicializa una fracción con el valor de 1
Fraccion::Fraccion()
{
   Num = 1;
   Den = 1;
}

// Constructor que inicializa una fracción con el valor dado por el usuario
Fraccion::Fraccion(int N, int D)
{
   Num = N;
   Den = D;
}

// Función que muestra la fracción en la forma acostumbrada
void Fraccion::Muestra()
{
   cout << Num << '/' << Den;
}
```

Se requiere un método llamado valorDecimal que calcule y regrese el valor decimal de la fracción dada. A continuación se muestra una versión para este método:

Dentro de la clase se declara el prototipo para el método:

```
Class Fracción {
   public:
      double Valor();     // Valor decimal de la fracción
      ...
};
```

Se define el método:

```
// Método que obtiene el valor decimal de la fracción
double Fracción:: Valor()
{
   return Num*1.0/Den;
}
```

Se manda llamar de la siguiente forma:

```
x = A.Valor();
```

Funciones Friend

Una función friend es una función que no es parte de la clase, pero puede ver la parte privada de la clase. Se especifica poniendo la palabra friend seguida del prototipo de la función dentro de la declaración de la clase.

Modifiquemos nuestro ejemplo para que la función Valor sea friend de la clase Fraccion.

Dentro de la clase se especifica el prototipo del método anteponiendo la palabra friend.

```
Class Fracción {
   friend double Valor(Fraccion f); // Valor decimal de la fracción
   ...
};
```

Se define la función libre; es decir, no es parte de la clase:

```
// Función friend que obtiene el valor decimal de la fracción
double Valor(Fraccion f)
{
   return f.Num*1.0/f.Den;
}
```

Se manda llamar de la siguiente forma

```
x = Valor(A);
```

Sobrecarga

Como ya se explicó antes, el lenguaje C++, permite que se definan funciones diferentes con el mismo nombre, para dar claridad a los programas, y evitarle al programador pensar en nombres diferentes cuando la funcionalidad de un módulo es la misma. El único requisito es que cada función tenga diferente cantidad de parámetros, o bien, parámetros de diferente tipo.

Sobrecarga de Operadores

Los operadores también pueden ser sobrecargados para que desempeñen funciones adicionales a las que tienen predefinidas. Esta funcionalidad tiene sentido cuando el significado de un operador se acopla a un nuevo tipo de dato (definido a través de una clase).

La sobrecarga se realiza al utilizar como nombre de la función la palabra operator y el símbolo del operador. Un operador que se sobrecarga tendrá la misma prioridad de ejecución y el mismo número de operandos que la del operador original.

Algunos de los operadores que se pueden sobrecargar se muestran enseguida:

ARITMETICOS: + , - , * , / , % , ++, --

RELACIONALES: < , > , <=, >= , == , !=

LOGICOS: && , || , !

OTROS: >> , <<

Utilicemos ahora el operador **!** para representar el valor decimal de una fracción.

Dentro de la clase se especifica el prototipo de la sobrecarga del operador anteponiendo la palabra friend.

```
class Fraccion
{
   friend double operator !(Fraccion f);
   ...
};
```

Se define la función libre; es decir, no es parte de la clase, nota que se cambia el nombre por operator !:

```
// Sobrecarga del operador unitario ! para obtener el valor decimal de la fracc
double operator !(Fraccion f)
{
   return f.Num*1.0/f.Den;
}
```

Se manda llamar de la siguiente forma:

```
x = !A;
```

Nota que al hacer la sobrecarga del operador solamente cambia el nombre de la función y la forma de llamarla, todo lo demás es igual. Además el significado original del operador se ignora, ahora tendrá el significado que le damos con la sobrecarga del operador.

Ejemplo completo:

```
// Ejemplo de la clase Fraccion, es para mostrar el uso de la sobrecarga
// de operadores que son friend o que no son friend pero no necesitan
// los datos de la clase (como es el caso del operador ++).
#include <iostream.h>

class Fraccion
{
   friend Fraccion operator + (Fraccion, Fraccion); // Suma
   friend Fraccion operator - (Fraccion, Fraccion); // Resta
   friend double Valor(Fraccion f); // Valor decimal de la fraccion
   friend int operator > (Fraccion, Fraccion); // Comparacion por >
   friend double operator !(Fraccion f); // Valor decimal de la fraccion
   public:
      Fraccion(int, int);
      Fraccion();
      void Muestra();
   private:
      int Num, Den;
};

// Constructor que inicializa una fraccion con el valor de 1
Fraccion::Fraccion()
{
   Num = 1;
   Den = 1;
}
```

```
// Constructor que inicializa una fraccion con el valor dado por el usuario
Fraccion::Fraccion(int N, int D)
{
   Num = N;
   Den = D;
}

// Funcion que muestra la fraccion en la forma acostumbrada
void Fraccion::Muestra()
{
   cout << Num << '/' << Den << endl;
}

// Sobrecarga del operador + binario (suma de fracciones)
Fraccion operator + (Fraccion f1, Fraccion f2)
{
   Fraccion res;

   res.Num = (f1.Num*f2.Den + f2.Num*f1.Den);
   res.Den = f1.Den*f2.Den;

   return res;
}

// Sobrecarga del operador - binario (resta de fracciones)
Fraccion operator - (Fraccion f1, Fraccion f2)
{
   Fraccion res;

   res.Num = (f1.Num*f2.Den - f2.Num*f1.Den);
   res.Den = f1.Den*f2.Den;

   return res;
}

// Sobrecarga del operador ++ (sumar 1 unidad a la fraccion)
// Esta sobrecarga NO esta definida como friend y NO es parte de la clase
Fraccion operator ++ (Fraccion f)
{
   Fraccion res, uno;

   res = f + uno;

   return res;
}

// Funcion friend que obtiene el valor decimal de la fraccion
double Valor(Fraccion f)
{
   return f.Num*1.0/f.Den;
}

// Sobrecarga del operador unitario ! para obtener el valor decimal de la fracc
double operator !(Fraccion f)
{
   return f.Num*1.0/f.Den;
```

```
}

// Sobrecarga del operador > (mayor que) regresa verdadero o falso
int operator > (Fraccion A, Fraccion B)
{
   if (!A > !B)
      return 1;
   else
      return 0;
}

int main()
{
   int x, y;

   cout <<"Teclea el valor de A (2,5)";
   cin>>x>>y;
   Fraccion A(x,y), B(3, 4), C;

   cout<<"A ";
   A.Muestra();
   cout<<"B ";
   B.Muestra();

   C = A + B;
   cout<< "Suma ";
   C.Muestra();

   C = A - B;
   cout << "Resta ";
   C.Muestra();

   C = ++B;
   cout << "Incremento ";
   C.Muestra();

   cout<<"Valor de la fraccion ";
   cout<<Valor(A);

   cout<<"\n Valor de la fraccion ";
   cout<< ! A;

   if (A > B)
      cout<<"\n A es mayor que B"<<endl;
   else
      cout<<"\n A no es mayor que B"<<endl;

   reutrn 0;
}
```

Ejercicio

Copia el ejemplo que se incluye en el material y pruébalo.

Un numero de la forma **a+bi**, donde **a** y **b** son constantes reales e **i** es la raíz cuadrada de -1, es llamado un numero complejo. El numero **a** es llamado la componente real y **b** es llamado la componente imaginaria.

La suma, diferencia y producto de dos números complejos son definidos por las siguientes ecuaciones.

Crea una clase para manejar números complejos, utiliza sobrecarga de operadores para programar las funciones de suma, resta y multiplicación de dos números complejos. Escribe una función main con la que puedas probar las funciones que creaste. ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Siguiente módulo

Objetos con arreglos y sobrecarga de operadores

Revisemos el siguiente ejemplo en el que se incluye una clase para representar conjuntos de números enteros. Recuerda que un conjunto es una colección de elementos en la que no hay elementos repetidos.

```
#include <iostream.h>

class Conjunto
{
      friend Conjunto operator +(Conjunto, Conjunto); //union de conjuntos
   public:
      int Pertenece(int valor);
      void PideDatos();
      void Muestra();
   private:
      int arr[20];
      int cant;
};

// Pide al usuario los elementos del conjunto, si se da un valor duplicado
// manda un mensaje de error
void Conjunto::PideDatos()
{
   int n, valor;

   cout<<"Cuantos valores hay en el conjunto ";
   cin>>n;
   cant = 0;
   while (cant < n)
   {
      cout<<"Teclea un elemento ";
      cin>>valor;
      if (Pertenece(valor) == 1)
         cout<<"Error, este elemento está duplicado"<<endl;
      else
      {
         arr[cant] = valor;
         cant++;
      }
   }
}

// Regresa 1 si el valor es elemento del conjunto y 0 si no lo es
int Conjunto::Pertenece(int valor)
{
   int encontro = 0;
   for (int i = 0; i < cant; i++)
      if (valor == arr[i])
```

```
        encontro = 1;
    return encontro;
}

// Muestra en la pantalla los elementos del conjunto
void Conjunto::Muestra()
{
    for (int i = 0; i < cant; i++)
        cout<<arr[i]<<endl;
}

// Funcion friend de la clase de los conjuntos (por eso puede usar los datos miembro
// de la clase de los conjuntos) que hace la union de dos conjuntos
Conjunto operator +(Conjunto A, Conjunto B)
{
    Conjunto Res;
    int i;

    // copio los valores del conjunto A al conjunto Res
    for (i=0; i<A.cant; i++)
        Res.arr[i] = A.arr[i];
    Res.cant = A.cant;

    // paso al conjunto Res los elementos del conjunto B que no estan en Res
    for (i=0; i<B.cant; i++)
        if (Res.Pertenece(B.arr[i]) == 0) // si el elemento no esta en el conjunto Res
            Res.arr[Res.cant++] = B.arr[i]; //nota que aquí se incrementa el contador cant de
Res
    return Res;
}

int main()
{
    Conjunto A, B, C;

    cout<<"Teclea los valores del conjunto A"<<endl;
    A.PideDatos();

    cout<<"Teclea los valores del conjunto B"<<endl;
    B.PideDatos();

    cout<<"La union de los conjuntos A y B es "<<endl;
    C = A + B;
    C.Muestra();

    return 0;
}
```

Ejercicio

Copia el ejemplo que se incluye en esta sección y pruébalo.

Agrega una función que realice la intersección de 2 conjuntos; sobrecarga el operador * para representar la intersección. Recuerda que la intersección de 2 conjuntos A y B es el conjunto

formado por todos los elementos que se encuentran en ambos conjuntos.

ver solución

Ligas sugeridas

http://www.cplusplus.com/doc/tutorial/

http://www.cs.wustl.edu/~schmidt/C++/

Regresar

Descripción del Proyecto Final

La tienda ?Computers Inc.? vende diferentes artículos de tecnología.

Haz un programa que muestre un menú con las siguientes opciones:

1. Dar de alta los artículos disponibles.
2. Hacer factura.
3. Ver el total de ventas hasta ese momento.
4. Hacer devoluciones
5. Salir del programa

Para dar de alta los artículos disponibles deberá

- Pedir la cantidad de artículos que vende la tienda; considera que como máximo venderá 30 artículos.
- Pedir la clave numérica, descripción (string), precio y cantidad en existencia de cada uno de los artículos.

Nota que esta opción debe utilizarse antes de poder hacer cualquiera de las otras, por lo que sería conveniente que el programa validara que se tenga que hacer primero esta opción.

Para hacer cada factura debe
Mostrar un mensaje que diga cual es el número de la factura (el programa debe asignar automáticamente el número)

Mostrar un encabezado como el siguiente:

Factura número 1 (número consecutivo, que aparece automáticamente)

Clave Articulo Descripción Cantidad Precio Unitario Precio

Y deberá permitir que el usuario teclee la clave del artículo y la cantidad que desea comprar. El programa mostrará automáticamente la descripción, el precio unitario y el precio total de ese artículo.

Después aparecerá en la parte de debajo de la pantalla un letrero que pregunte si se desea comprar otro artículo; si el usuario responde que si, el programa debe repetir el proceso de pedir la clave y cantidad del artículo a comprar en el siguiente renglón de la pantalla; similar a como se muestra en la siguiente figura:

Factura número 1

Clave Articulo	Descripción	Cantidad	Precio Unitario	Precio
234	Monitor	2	238.00	476
375	Teclado	2	250.00	500

¿desea comprar otro artículo (s/n)? _

Cuando el cliente diga que no quiere otro artículo el programa deberá mostrar el total de esta factura y volver al menú. Considera que como máximo se venden 10 artículos por factura.

Para ver el total de ventas hasta ese momento

Se mostrará un listado de los números de factura con sus saldos y el total de ventas hasta ese momento.

El programa debe dar la opción de ver a detalle una factura, esto se hará preguntando en esta pantalla el número de factura y después mostrando su contenido.

Devoluciones

El programa deberá pedir el número de factura, la clave del artículo y la cantidad de artículos a devolver.

Se deberá registrar la devolución en la factura correspondiente; es decir modificar el contenido de la factura para reflejar los artículos comprados (quitando solo los que se devolvieron) y se calcule de nuevo el total de la factura.

Es importante que los artículos a devolver estén de nuevo disponibles para su venta.

Una vez realizado el proceso debe de enviar un mensaje de confirmación de la devolución.

Notas Importantes

El programa deberá validar todos los datos que pida al usuario; es decir:

- Que no se puedan hacer facturas sin haber dado de alta el inventario.
- Que no se puedan registrar más de 30 artículos en la tienda.
- Que no acepte cantidades negativas para precios, cantidades a comprar, claves, números de factura, cantidad a devolver, etc.
- Que la clave artículo sea un número existente.
- Que no se puedan vender más artículos de los que hay en existencia.
- Que al hacer devoluciones el número de factura sea válido y que la clave del artículo corresponda a un artículo que existe en esa factura.
- Que no se teclee una opción inválida en el menú.
- Que cuando se pregunte por si o no verifique que solo se teclee s o n en mayúsculas o minúsculas.

Regresar

Baja el compilador de Borland turboC++ en la dirección:
http://www.borland.com/products/downloads/download_cbuilder.html#

Baja el editor de Crimson en la siguiente dirección:
http://www.crimsoneditor.com/english/download.html

Instala el compilador dando doble clic al archivo de instalación:
freecommandLinetools.exe

Baja de los recursos los archivos BCC.BAT y RUN.BAT y copialos en la ruta
c:\Borland\BCC55\Bin

Instala el editor Crimson dando doble clic al archivo de instalación
crimsoneditor.exe

Configura el editor de Crimson usando los siguientes pasos:

- Ejecuta el editor, al hacerlo por primera vez aparece un mensaje de error, solo da Aceptar
- En el Menú ***Tools***, selecciona ***Preferences***
- En la ventana de Categorías, selecciona ***Syntax*** y elije ***C/C++***
- En la misma ventana, selecciona ahora ***User Tools*** y has clic donde dice ***Empty Ctrl 1*** y llena los recuadros de la siguiente forma:
  Menú Text : Compilar (puedes poner cualquier texto, es solo el mensaje que aparecerá en la ventana de Tools, que indica la opción de compilar el programa).
  Command: *c: \borland\bcc55\bin\bcc.bat*
  Argument: haciendo clic en la flecha selecciona *File Name*
  Initial dir: haciendo clic en la flecha selecciona *File Dir*
  Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.

  Ahora has clic donde dice ***Empty Ctrl 2*** y llena los recuadros de la siguiente forma:
  Menú Text : Ejecutar (puedes poner cualquier texto, es solo el mensaje que aparecerá en la ventana de Tools, que indica la opción de ejecutar el programa).
  Command: *c: \borland\bcc55\bin\run.bat*
  Argument: haciendo clic en la flecha selecciona *File Title*
  Initial dir: haciendo clic en la flecha selecciona *File Dir*
  Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.

  Por útlimo da Apply.

Prueba cuidadosamente los ejemplos, revisando que sucede con diferentes valores.

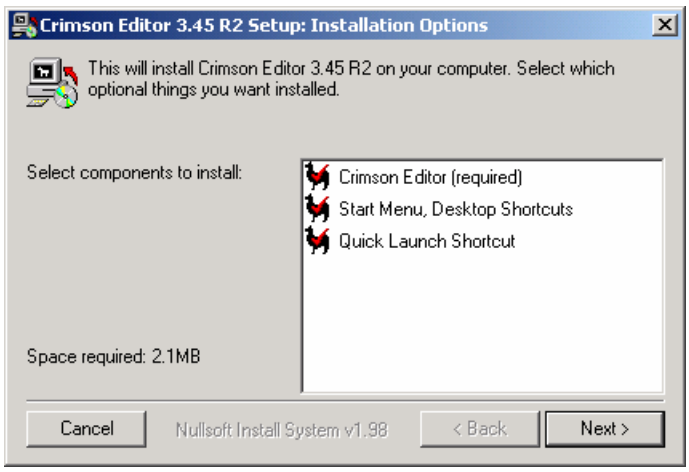
Instalando el Crimson Editor©

Descarga la ultima versión disponible (aproximadamente 850 kb) de la siguiente página
http://www.crimsoneditor.com/english/download.html

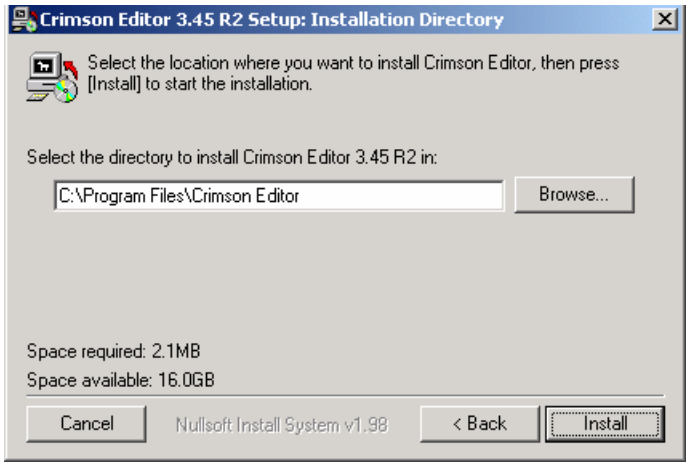
Requerimientos del Sistema:
Windows 95, Windows NT 4.0 o superior
3 MB de espacio libre en el disco duro.

PASO 1:
Doble clic sobre el ícono de la instalación:

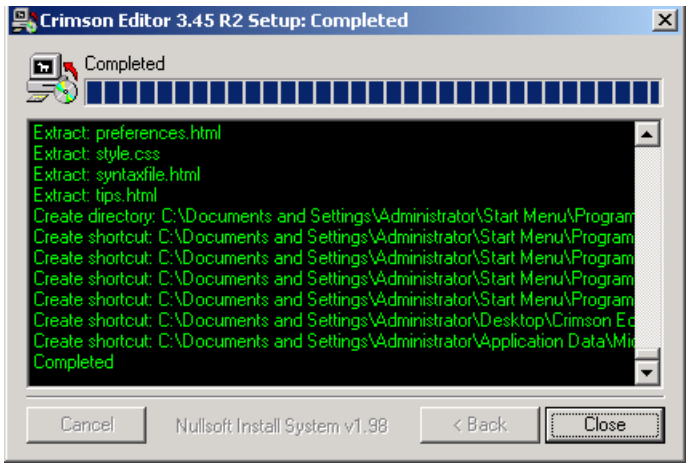
PASO 2:
Aparecerá la siguiente ventana. Haga clic en **Next**.



PASO 3:
En la siguiente ventana, haga clic en **Install**.



PASO 4:
Al terminar la instalación, haga clic en **Close**.



PASO 5:

En el *desktop* encontrará el ícono del editor Crimson. Haga doble clic sobre él para abrir el Crimson con el fin de configurarlo para que llame al compilador de Borland C++.

Otra forma de ejecutar el editor Crimson es a través del menú "**Inicio**", dentro de la sección "**Programas**", en el grupo "**Crimson Editor**".

Configurando el Crimson Editor© para programar en C++

NOTA:
Esta guía asume que:

- Ya se ha instalado el compilador Borland C++ 5.5
- Ya se han colocado los archivos BCC.BAT y RUN.BAT en la carpeta llamada C:\Borland\BCC55\Bin\.

  El contenido del archivo BCC.BAT es el siguiente:

```
@echo off
path=c:\borland\bcc55\bin
BCC32 -LC:\BORLAND\BCC55\LIB -IC:\BORLAND\BCC55\INCLUDE %1
```

  El contenido del archivo RUN.BAT es el siguiente:

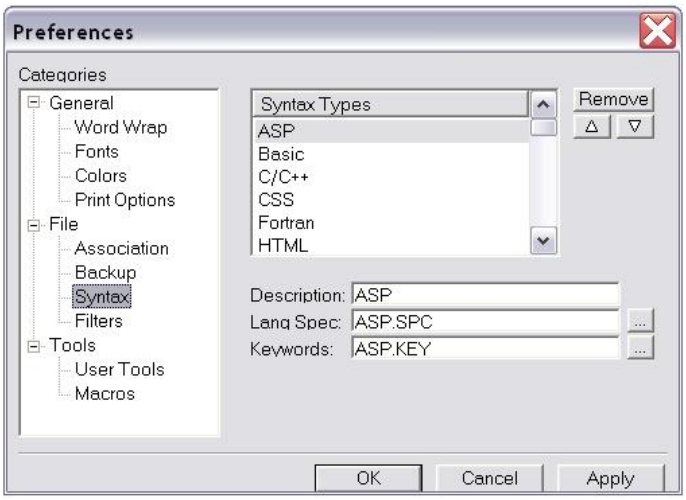
```
%1.exe
```

PASO 1:

Al ejecutar por primera vez el editor, aparecerá el siguiente mensaje de error. Haga clic en el botón **Aceptar**.

PASO 2:

En el menú **Tools**, seleccione **Preferences**.
En la ventana que aparecerá, dentro de la sección **Syntax Types**, elija **"C/C++"**.



PASO 3: (En este paso se instalar un comando para llamar al compilador de C++)
Estando aún dentro de **Preferences**, haga clic en la sección **Tools -> User Tools**.

Haga clic en donde dice "**Empty Ctrl+1"** y llene los siguientes datos:

Menu Text: Compilar el programa (puede ser cualquier texto)

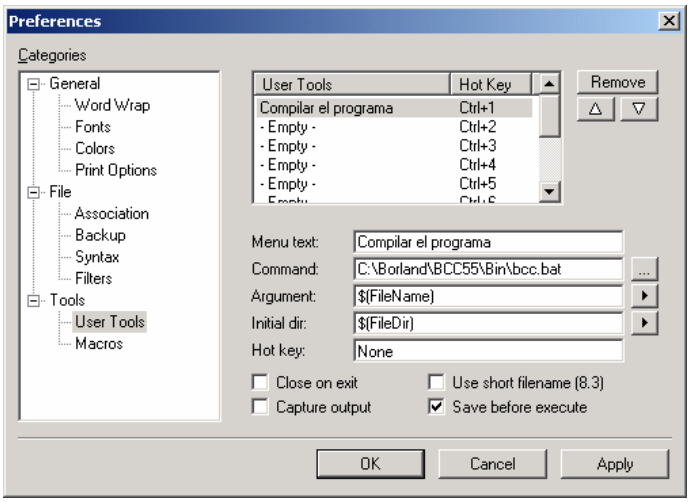
Command: C:\Borland\BCC55\Bin\bcc.bat

Argument: Haga clic en la flecha de la derecha y seleccione **File Name** $(FileName). Se llenará el campo automáticamente.

Initial dir: Seleccione **File Directory** $(FileDir).

Hot key: Se puede dejar sin modificar, o puede presionar la secuencia de teclas que "ejecuten" este comando. Por *default* el programa asigna la combinación de teclas Ctrl+1 para el primer elemento del menú de **User Tools**, las teclas Ctrl+2 para el segundo, y así sucesivamente.

Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.



PASO 4: (En este paso se instala un comando para ejecutar el programa de en C++)

Haga clic en donde dice "**Empty Ctrl+2"** y llene los siguientes datos:

Menu Text: Ejecutar el programa

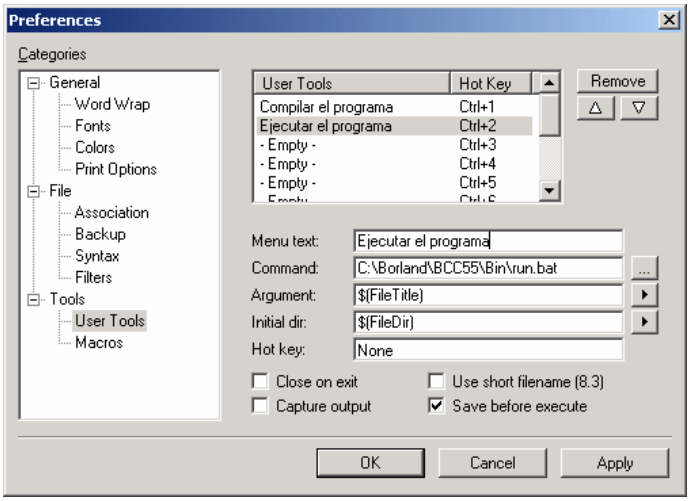
Command: C:\Borland\BCC55\Bin\run.bat

Argument: **File Title** $(FileTitle)

Initial dir: **File Directory** $(FileDir)

Las opciones **Close on exit**, **Capture output** y **Use shor filename** se dejan sin seleccionar. La opción **Save before execute** queda seleccionada.

Haga clic en el botón **Apply**.
Cierre la ventana de **Preferences**.

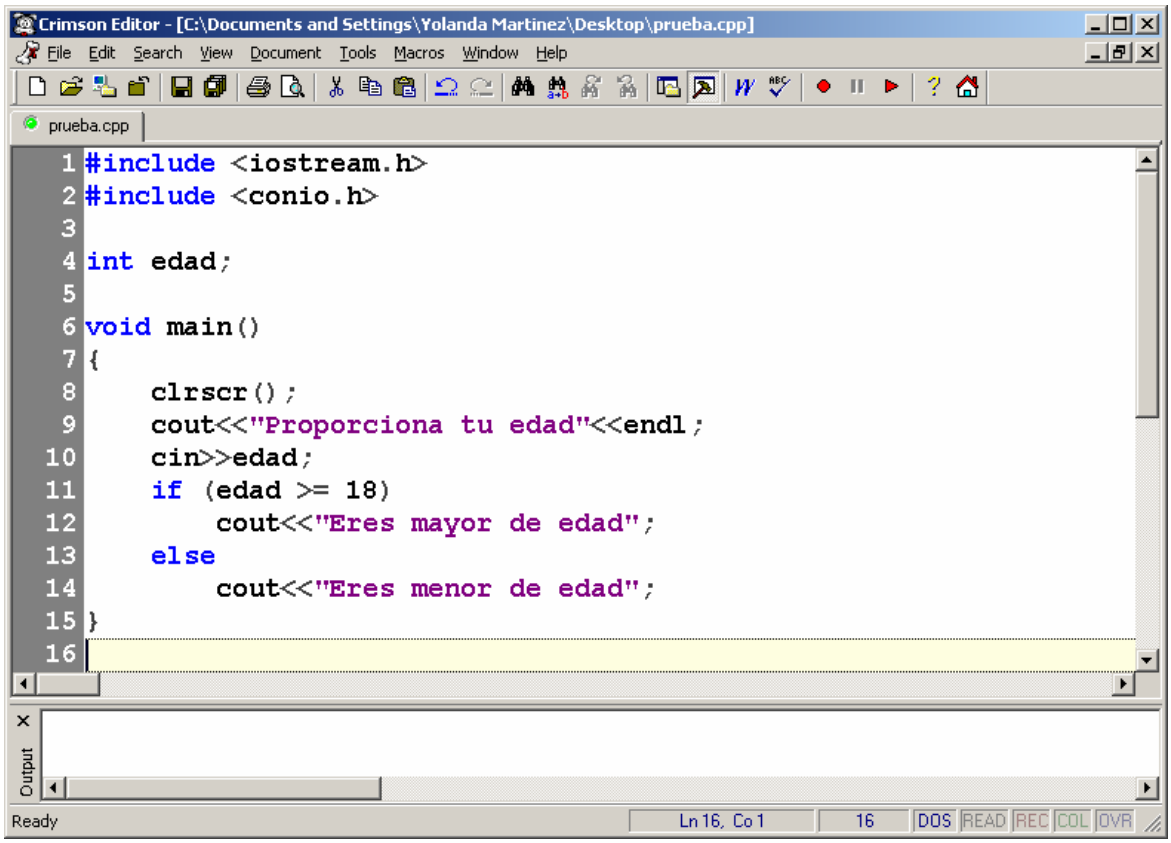


Con estos pasos ha quedado configurado el editor Crimson para llamar al compilador de C++ y ejecutar el programa generado.

Ejemplo:
Creando y ejecutando un programa de C++ dentro del Crimson Editor©

PASO 1:

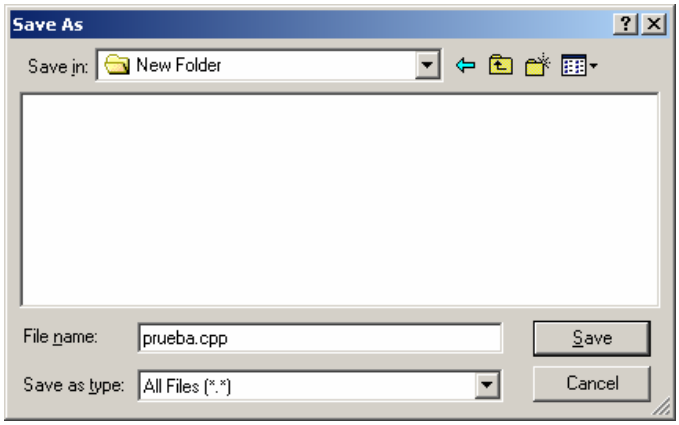
Abra el editor Crimson y escriba el siguiente programa.



PASO 2:

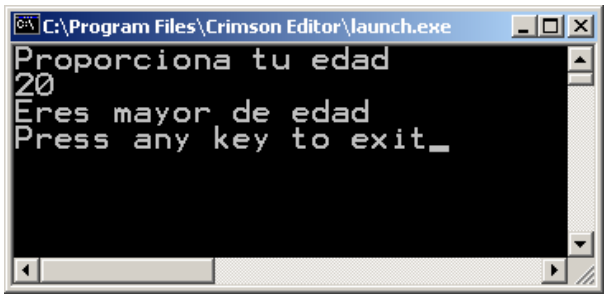
Salve el programa con el nombre “prueba.cpp”. En el menú **File**, elija **Save**.

File name: prueba.cpp
Save as type: C/C++ Files (*.c,*.cc,*.cpp,*.cxx,*.pc,*.h)



PASO 3:

Para verificar que el programa escrito esté libre de errores hay que compilarlo. En el menú **Tools**, elija la opción **Compilar el programa**. Se abrirá una ventana, en la cual se describen los errores encontrados (en caso de que existan). Al terminar, indica “Press any key to exit”, para que el usuario presione cualquier tecla para terminar y proceder a corregir los errores o a ejecutar el programa (si no hubo errores).

PASO 4:

Si no hubo errores en la compilación, se generó el archivo cuatro.exe (de este ejemplo), el cual es el archivo ejecutable. Para ejecutarlo, en el menú **Tools**, elija la opción **Ejecutar el programa**.